2011

Complete coverage path planning in an agricultural environment

Theresa Marie Driscoll
Iowa State University

Complete coverage path planning in an agricultural environment

By

Theresa Marie Driscoll

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Science

Program of Study Committee:
Yan-Bin Jia, Major Professor
David Fernandez-Baca
Jennifer Davidson
Lie Tang

Iowa State University

Ames, Iowa

2011

Dedication

To my husband for taking care of the children while I was studying, and for putting up with my late hours working on this thesis.

Table of Contents

List of tables

List of figures

Abstract

The problem of finding a collision-free path through a region has garnered a lot of research over the years. One branch of this is the problem of finding a path that completely covers a region. Solutions to the complete coverage path planning problem have applications in many different areas, such as search and rescue, automotive painting, and agriculture. In many cases, it is not sufficient to find any route that completely covers the field. It is desired that the path also be optimal so as to minimize certain costs. This is especially true in the agricultural environment. In the area of precision farming alone, the complete coverage path planning problem exists while performing many different operations, such as harvesting, seeding, spraying, applying fertilizer, and tillage. The fundamental concern of farmers is reducing the costs of running the farm. Since most farming costs ultimately depend on time in the field and area covered, the more efficient an operation can be completed, the lower the costs. Optimality is thus typically in terms of finding the shortest complete coverage path through the field. In this paper, we present an $O(n^2)$ algorithm for solving the optimal complete coverage problem on a field boundary with n sides. This multi-phase algorithm makes use of a plane-sweep algorithm to subdivide the field into smaller, trapezoidal regions. The optimal paths through the subregions are then calculated. Finally there is a merge phase where it is determined whether neighboring regions can be more efficiently covered if they were merged together than if they were left separate.

CHAPTER 1: INTRODUCTION

1.1 Problem Statement

In farming operations, the farmer's ultimate goal is to make the most use of the land so as to bring in the most income. Over time, farming costs have increased drastically. Not only are there the costs of seed and fertilizer, but also fuel costs and salaries of hired hands. As smaller farms have consolidated into larger farms, farm owners require more hired hands. Often times the hired hands were not raised on farms and do not have the same level of knowledge or experience as the farm owners. Automating tasks and planning operations out in advance helps to reduce these costs and eliminate mistakes. Automated solutions need to know or determine several things: what is the task to be done, what are the steps to complete it, where is the task to be performed, where is the machine or implement currently located, what other machines or implements are in the field that need to be coordinated with, and what is the path through the field to be taken. The task to be done and its steps are typically conveyed to the automated solution as input. Global positioning systems (GPS) typically relay the current location of the machine or implement to the algorithm. The remaining two problems to be solved are how to coordinate multiple machines and implements (mission planning) and how to determine the optimal path through the field (complete path planning). This research focuses on the latter problem.

To maximize profits, it is obvious that as much of the field as possible should be put to use. Ideally, any path through the field should attain 100% coverage. But the complete coverage path planning problem is not as simple as finding any path that completely covers the region. As years have gone by and family farms have gotten bigger and started competing with corporate farms, and prices have gotten higher, another concern is the cost of farming the land. In order to even make a profit, costs need to be kept as low as possible. In farming operations, costs come in many forms, including cost of labor, cost of fertilizer, chemicals or seed used, gas usage, and time. Ultimately, many these costs can be expressed in terms of time in the field – the more efficient the farming operation, the better chance of earning profits. As a result, it is not

sufficient just to find a path that completely covers the field – it must be an efficient one. Thus a second goal of the path planning problem is finding a path that is as optimal as possible.

Finally, when applying fertilizer or chemicals, or planting, double coverage should be avoided. In such operations, paths through the field should not overlap. In addition, regardless of the operation, there are certain regions of the farm that for one reason or another are not passable. Further, there may be obstacles such as trees in or along side of the field. So to be usable in the agricultural environment, any proposed path should not collide with obstacles or enter impassable regions.

To summarize, any solution to the complete coverage path planning problem in an agricultural environment needs to satisfy the following requirements:

1. Cover as close to 100% of the land as possible.
2. Avoid double coverage of areas.
3. Avoid obstacles and impassable areas.
4. Be as efficient as possible. Keeps costs to a minimum.

1.2 Prior Research

There are several different approaches toward solving the collision-free path planning problems in general, and the complete coverage path planning specifically. At a high level, these approaches can be categorized into two groups: “global” and “local”. Global approaches look to the workspace as a whole, building a model of the entire workspace and the location of obstacles, and searching this workspace for the optimal solution. Global approaches are typically used when the environment is known in advance, and generally are not on-line algorithms. In contrast, the local approach generally involves gridding the work space and searching cells local to the robot’s current position. Heuristics computed from the area seen so far helps guide the robot. As a result, the local approach is typically used when the environment is not known in advance, and generally are on-line algorithms that require the use of sensors. Both approaches involve subdividing the workspace in some fashion, either by gridding it or by partitioning the area into

smaller polygonal regions. Various techniques have been devised for both approaches based on the tools available from the fields of computer science and mathematics. These techniques include cellular decomposition, artificial potential fields, neural networks, genetic algorithms and algorithms for solving the traveling salesman problem.

One approach typically used in global approaches is cellular decomposition. This involves subdividing the region into subregions (typically trapezoids or triangles, but it could be any shape), find the best way to cover each subregion, and then aggregate the results. Choset and Pignon (1997), Choset (2000), and Atkar et al. (2001) were among the first group of researchers to propose combining exact cellular decomposition with boustrophedon (back and forth straight passes in alternating directions) to automatically find a path that completely covers a region. Their approach involved a modified plane-sweep algorithm to decompose the area into subregions. Instead of treating all points as "events" where a subdivision is to occur, subdivision only occurs at "in" events and "out" events. At an in event, or split point, the current cell (if any) is closed and two cells are opened. At an out event, or merge point, two cells are closed and one cell is opened (assuming it's not the last point on the outer boundary). At all other points, the shape of the cell is simply updated. Figure 1 compares this type of decomposition with standard trapezoidal decomposition. Each subregion was then covered using back and forth paths. The order in which the regions were visited was computed using a depth first search, marking each cell as visited until all cells are covered. The choice of the sweep direction can affect the optimality of the solution. However, this group of researchers did not seek to find the optimal solution – only finding a feasible complete coverage solution. As such, the sweep direction was chosen at random.

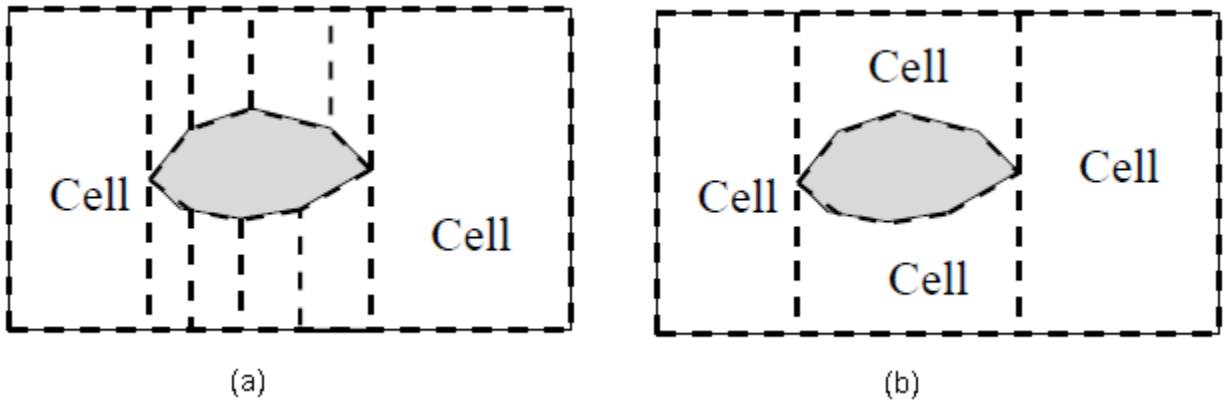


Figure 1. Comparison of trapezoidal decomposition with boustrophedon decomposition Choset (2000). (a) Trapezoidal decomposition. (b) Boustrophedon decomposition

Oksanen and Visala (2007) extended the exact cellular decomposition approach to the agricultural environment, and to take into account the costs of traveling in a particular direction. Oksanen and Visala formulated an iterative subdivide/merge algorithm. The field was first subdivided using the standard plane-sweep algorithm to decompose the area into trapezoids. Then the subregions were merged into as large as regions as possible, while retaining relatively rectangular shapes. The "best" region was then broken off, and traveled in the same direction as the parallel sides of the trapezoids. The algorithm repeated on the remaining subregions until the entire area was covered. The "best" region was determined by a weighted sum of three factors: area, distance traveled, and turning times. In each iteration, the direction to cover the field in boustrophedon paths was calculated by computing costs on the un-subdivided region. The sweep direction was set to be normal to the calculated optimal direction. The cost calculation was unspecified, but to speed up running time, costs were calculated in one of six directions: 0°, 30°, 60°, 90°, 120° and 150°. The best three directions were chosen, the step size (initially 30) was halved, and new search directions were added on each side of the original three. This repeated for five iterations to recursively narrow down on the best direction without actually calculating cost in all directions. Although they did not discuss the running time of this algorithm, given that the plane sweep algorithm runs in O(n log(n)) time and produces O(n) trapezoids, and the merge algorithm can be implemented to run in O(n) time, it has an overall running time of $O(n^2 \log(n))$. This algorithm suffers from the problem Choset and Pignon (1997) were trying to avoid: too

many subregions processed separately with potential overlaps on the shared edges or the need for headlands to avoid the overlaps.

Jian Jin, a PhD graduate of Iowa State's Agricultural and Biosystems Engineering, also conducted research that fell in this area. Jin and Tang (2006) and Jin (2009). His approach was to find "all possible ways" for splitting the field into two, and then to try each of the possible ways to see if the region could be more efficiently as two separate regions instead of one. The algorithm continued recursively on each subregion until it was no longer possible to split the region to achieve a more optimal solution. The algorithm had a running time of $O(n^3 \log n)$. Although it yields reasonable solutions, the algorithm suffers from performance. It runs reasonably well in fields with no interior obstacles, but has a significant performance hit when interior obstacles are added to fields with more than twenty points.

Others have limited the search space or considered alternative methods of sweeping besides boustrophedon paths. Kang et al, (2007) limited the search space by rotating the field so that the most dominant line component of the boundary is horizontal, then doing line sweeps left to right to find the critical points for starting and stopping the sub fields. With the field thus rotated, they considered each of three groups of motions to find the optimal direction – left to right passes, up and down passes and spiral passes. Of each group, four different passes were considered, one for starting in the upper right corner, one for starting in the lower right corner, one for starting in the lower left corner and one for starting in the upper left corner of the sub field.

Gonzalez et al. (2005) proposed an on-line complete coverage algorithm that uses spiral filling paths instead of back and forth paths. The algorithm subdivides the workspace into simple regions as the area is covered. A region that can be (and is) completely covered using a single spiral path is called a simple region. The algorithm starts from a grid of the workspace where all cells are marked as unknown. As the robot moves, it detects whether obstacles are to the front or one side of it, and updates the environment model and makes a move. As cells are entered, they are marked as "virtual obstacles" so that the same area is not covered more than once. Once a simple region is covered, a backtracking algorithm is applied to determine the next region to go to. The algorithm assures complete coverage of non-occupied cells, but not of partially occupied

cells. The basic version of the algorithm attains coverage generally in the 62-67% range. Even modified with a wall following algorithm, coverage is around 92-93%.

Another approach to solving the path planning problem is to make use of artificial potential fields. The main idea is to construct an attractive force at the destination and a repulsive force on the obstacles, then let the forces "pull" the robot to the destination. Barraquand et al. (1992) were one of the first to use potential fields to navigate a robotic arm around an environment. Zelinsky et al. (1993) extended the use of artificial potential fields to the complete coverage situation. The workspace is gridded and a "start" and "goal" location is picked. Instead of propagating just a distance from the goal wave front through the free space, a wave front that is a weighted sum of the distance from goal and distance to an obstacle is propagated. This approach does not take into account the constraints of the vehicle, such as its turning radius, or other costs, such as turning costs, and can yield results that are impossible for farming equipment to follow. Sörensen (2003) developed an artificial potential field algorithm to steer a non-holonomic vehicle along crop rows without damaging the crops. Potential fields tend to suffer from getting trapped in local minima, and for the most part have only been applied to the point A to point B path planning problems.

Others have looked to the field of artificial intelligence for ways to solve the path planning problem in a static environment. Research exists in the area of neural networks (both learning and non-learning), genetic algorithms, ant colony algorithms and intelligent water drops algorithms. Most of the research using neural networks was done in the mid to late 1990's. See e.g. Martin and Del Pobil (1994). For example, Tse et al. (1998) developed a back propagation neural network which could generate a robot path. During the cleaning process, the robot recorded the previously generated path, the number of grid points traveled, and the number of turns. If something unexpected in the environment appears, the memory map had to be updated. In more recent years, Yang and Luo (2004) developed a non-learning neural network which was capable of generating collision free complete coverage paths in a non-stationary and unstructured environment. Working much like an artificial potential field, each neuron represents an area – typically a grid cell – in the workspace. The dynamics of the *i*'th neuron is characterized by a shunting equation derived from Hodgkin and Huxley's (1952) membrane equation and constructed so that the robot is globally attracted to uncovered opened areas and locally repelled

from obstacles. The network is configured so that the positive neural activity propagates to the entire space and the negative activity stays locally. The running time of the algorithm on a workspace N x N square was $O(N^2)$ – i.e. squarely proportional to the resolution of the grid. It's worth noting that the paths generated tended to be boustrophedon paths. Guo and Balakrishnan (2006) extended on this work to take into account the physical robot's kinematic motion constraints, and to generate smooth trajectories using parametric polynomials.

More recent research use neural networks only for steering the vehicle, using some other technique for determining the optimal path. Noguchi and Terao (1997) were one of the first to combine neural networks with genetic algorithms to plan the work path of an agricultural mobile robot through a field. They used the neural network to describe the motion of the vehicle – to essentially steer it – and the genetic algorithm to determine the path it would take through the field. More recently, Du, Chen and Gu (2005) combined neural networks and genetic algorithms by using neural networks to model the environment and a genetic algorithm to find a path from point A to point B through the environment. Neuron C_i^k outputs 1 if the point (X_i, Y^i) is in the k'th obstacle, and outputs 0 otherwise. The set of possible points from point A to point B were then flattened to one dimensional space so that each chromosome would represent the path in the flattened space. The fitness of each proposed path is based on whether it intersects an obstacle and is as short as possible.

Noting that prior research only used genetic algorithms to find a Point A to Point B path through an environment, Ryerson and Zhang (2007) investigated the feasibility of using genetic algorithms to solve complete coverage path planning problem in an agricultural environment. Each chromosome was a list of 2D points. The environment was modeled as a grid, with the assumption that if the vehicle covered the center of the grid, everything in that grid cell would be covered. Fitness was based on a combination of whether it covered 100% of the area, whether it intersected obstacles, whether it crossed over itself and the length of the path. Subsequent generations were created using a combination of crossover, mutation, reordering of points, swapping points and removal of crossings. Ryerson and Zhang were able to attain approximately 90% coverage, but not the most efficient path.

Yet another approach commonly taken to solve the complete coverage path planning problem is to translate the problem to a traveling salesman problem. This generally involves gridding the region into cells such that if the centers of all the cells are visited, 100% coverage would be reached. Thus viewed, any algorithm that can solve the traveling salesman problem could also solve the complete coverage path planning problem. Among such algorithms are swarm based intelligence algorithms such as the ant colony algorithm, Dorigo and Gambardella (1997), and intelligent water drops algorithm, Shah-Hosseini (2009). In the ant colony algorithm, *m* "ants" are initially placed on randomly selected locations. At each time step, they move to a new location, modifying the "pheromone trail" on edges used. Ants will tend to follow trails with higher pheromone levels. A tabu list is maintained to track which locations have already been visited so that the ants do not revisit that location. When all ants have completed a tour, the ant that made the shortest tour deposits more pheromones along the route that ant took. Then the algorithm repeats until the desired threshold is reached. Similar to the ant colony algorithm, the intelligent water drops algorithm models the properties and behavior of water as it flows downhill – namely that it prefers straight paths, only veering from straight paths if obstacles exist, that it prefers paths with low soil, that it tries to modify its environment by relocating soil, and that it has a speed that increases or decreases depending on the soil relocated.

Although intriguing algorithms, swarm-based algorithms can take a large number of iterations before it converges on a solution, and the algorithm can be trapped in a local minimum. Chibin, Xingsong, and Yong (2008) attempted to reduce the size of the traveling salesman problem by combining it with cellular decomposition approaches. The workspace was first decomposed into regions as in Choset (1997, 2000). Each region would be covered using boustrophedon paths. The order in which the regions would be processed was then determined using the ant colony algorithm. A distance matrix was then computed from the connectivity graph of the subdivided regions and used to compute the total distance traveled.

Finally, Meuth and Wunsch (2001) presented an approach that combines cellular decomposition, a solution to the traveling salesman problem, and genetic algorithms. It used the divide and conquer methods from cellular decomposition first to subdivide the field. Then for each sub field, it used a modified version of the Lin-Kernighan algorithm for the travelling salesman problem. The Lin-Kernighan algorithm was modified so that instead of randomly

generating another tour in each iteration, the modified version used a genetic algorithm to generate the next tour for the next iteration. As with some of the other algorithms mentioned above, this approach resulted in paths with many turns. This was true even when the cost function was modified to take into account the mobility of the vehicle. Further, as with other genetic algorithm approaches, convergence on a near optimal solution is not guaranteed.

1.3 Research Direction

Considering all the previous research in this area, the research in the area of cellular decomposition seems like it would have the most promise to efficiently yield a close to optimal solution in an agricultural environment. In the agricultural scenario, the field boundaries and obstacle locations are generally well known in advance, and as such, there is less of a need for an on-line implementation, and less of a need to learn the environment. The goal of this research is to find an $O(n^2)$ or better algorithm that finds a reasonably efficient solution to the problem of completely covering a given field. To accomplish this, this research aims to:

1. Make use of the research of Jin (2009) as to the cost functions and headland types.
2. Build upon the work of Jin (2009) so as to improve the performance of the algorithm while achieving similar or better results. Whereas Jin (2009) searched in all possible directions for ways to subdivide the field, the search space is narrowed by sweeping in the direction normal to the overall optimal direction and performing a modified trapezoidal decomposition. Doing so, hopefully the increase in performance will not come at the expense of giving solutions that are far from optimal.
3. Build upon the work of Oksanen (2007) by combining his subdivide and merge phases into a single subdivide phase, and by including a new merge phase that takes into account cost savings while merging. This should reduce the number of subregions that are separately covered, resulting in less double-coverage and fewer headland usages, and slightly improve the overall running time. Whereas Oksanen (2007) iteratively applied a subdivide/merge/split off approach, the subdivision and merge will be done only once.
4. Analyze the results of running this algorithm on various real world fields.

CHAPTER 2: COST FORMULATION

2.1 Cost Functions in Prior Work

In the previous work, there have been different formulations of the cost function. Some simply counted the number of turns – trying to keep the number of turns to a minimum. In the TSP/genetic algorithm hybrid approach, the cost function was in terms of time: (Euclidean distance of a straight path / speed) + (speed / vehicles acceleration capability) * vehicle agility * (1 – cos(θ)). Meuth and Wunsch (2001) defined the vehicle agility as its ability to execute turns without incurring extra time. So the above formula basically calculates the time to make a straight pass plus the time in the turns.

Likewise, Jin (2007) used the time taken as the cost function. However, he focused exclusively on the time turning in the headland, noting that time inside the field is relatively constant. He had three parts to his cost function: the cost of straight paths to clear the boundary between the headland and the inside of the field, one for the straight movement in the headlands before actually turning, and one for turns in the headlands. He assigned different cost coefficients (time / distance) for each of the three parts. In his PhD thesis, Jin (2009) thoroughly categorized the different turn types, determined the scenarios in which each turn type applied, and presented distance formulas for each turn type. He also considered the effect headland width had on the turning cost.

Technically, Oksanen (2007) did not specify a cost function for determining the optimal sweep direction at the beginning of each iteration of his algorithm. However, he did specify the selection criteria for selecting the “best” subregion to break off the field after the sweep and merge phases. He avoided a selection criteria based solely on the number of turns or efficiency, noting that such a selection criteria tended to be biased towards directions parallel to the longest direction, and may lead to final decompositions with many long and narrow cells, when other ways of decomposing the field may be more cost efficient. To avoid this result, Oksanen (2007) took into account three factors: area covered, distance traveled inside the field and overall

efficiency, where efficiency was based on distance traveled in the field plus the time spent in the headlands. He gave relative weights of 65%, 15% and 20% respectively.

In genetic algorithm approaches, fitness scores are the closest calculations to a cost function. In her genetic algorithm approach, Ryerson (2007) had a fitness score based on four factors: whether the path hit obstacles, whether it was as short as possible, whether it crossed over itself and whether it covered as much an area as possible, with different weights given to each factor. In their neural network/genetic algorithm hybrid approach, Du et al. (2005) based their fitness function solely on whether the path intersected itself and was as short as possible. Complete coverage was not a concern in that scenario.

2.2 Preliminary Cost Formulation Findings

As previously mentioned, Jin (2009) only considered distance along turns in the cost function, noting that in-field distance traveled is roughly the same because the area and pass width is fixed. Eliminating consideration of the in-field distance would be ideal as it considerably simplifies the calculation of the cost function. If in-field distance was included in the cost function, then for each pass through the field, it would need to be determined which edges of the boundaries are the opposing ends of the pass. Making that determination for each pass obviously would drastically slow down the algorithm. Yet, distance in field can vary depending on the placement of the first pass, the ratio of the pass width to the length of an edge, and whether a partial pass is needed to complete the field. Before fixing on a cost function that did not take into account in-field distance traveled, two questions therefore needed to be answered: (1) how much of an impact does in-field distance have on the overall cost calculation, and (2) can it safely be ignored from the cost calculation.

To see how much the in-field distance can vary and whether the variance has a significant effect on the overall result, several experiments were run on polygons of varying sizes and pass widths. Total distance traveled along boustrophedon paths in a given direction was evaluated at every quarter degree on the range 0 to π. The general theme noticed is the plot of the total distance traveled (in-field plus turning length) pretty closely tracks the curve when considering

distance in turns only. Although the overall optimal direction does not necessarily match exactly with the optimal direction if only distance in turns were considered, the difference in distance traveled in the overall optimal direction and the optimal direction when considering turns only is relatively small.

Consider the following figures plotting the various minimum distances. Figure 2 depicts two example polygonal shapes evaluated for their optimal directions. Throughout this thesis, a given path direction is represented by θ, and is measured counterclockwise from horizontal.

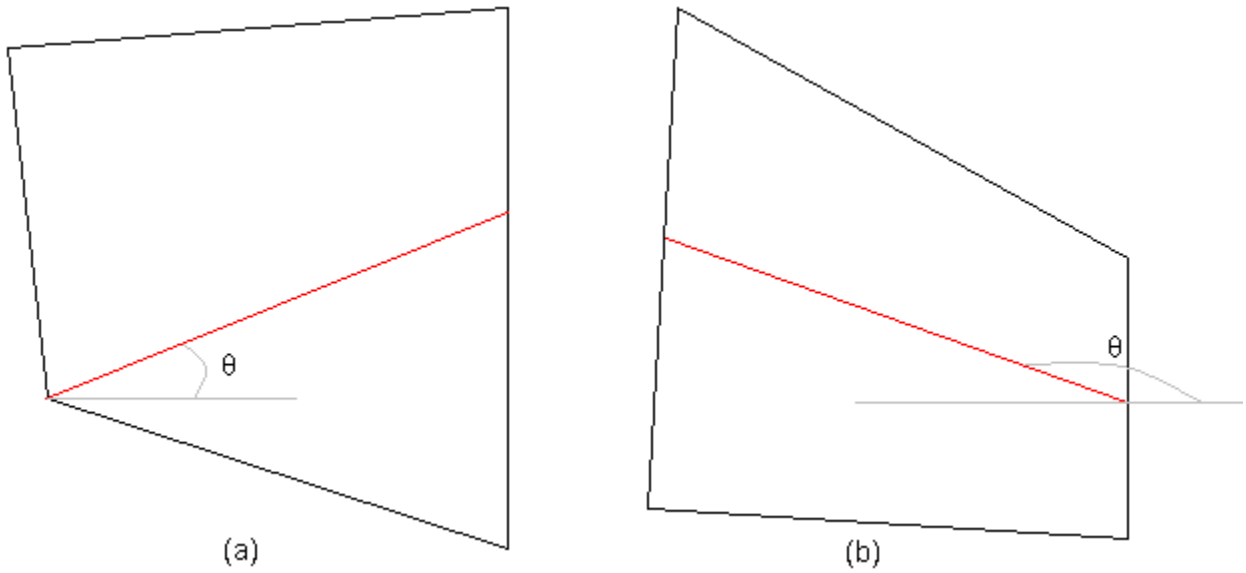


Figure 2. Sample polygons.

Figures 3 and 4 plot the distance traveled, both in-field and on turns, when the pass width is set to 10. In these plots, the vertical axis represents length and the horizontal axis represents the angle at which the length in field, turning distance and total distance traveled was computed. Notice from both these figures how the plot of the total distance traveled tracks the plot of the distance traveled in turns. These experiments were run on trapezoids, four sided polygons, and multi-sided polygons, and the results depicted here are typical for all polygons considered.

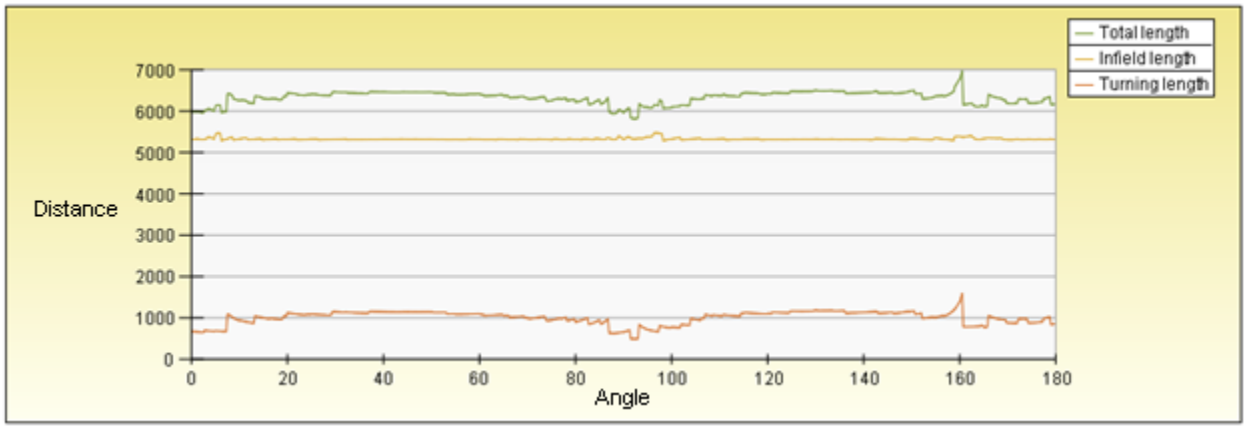


Figure 3. Plot of distance traveled in polygon A when the pass width is 10.

Figure 4. Plot of distance traveled in polygon B when the pass width is 10.

Also investigated was whether these results hold true regardless of pass width. The same calculations were performed while changing the pass width. Figures 5 and 6 depict the results from two different scenarios.

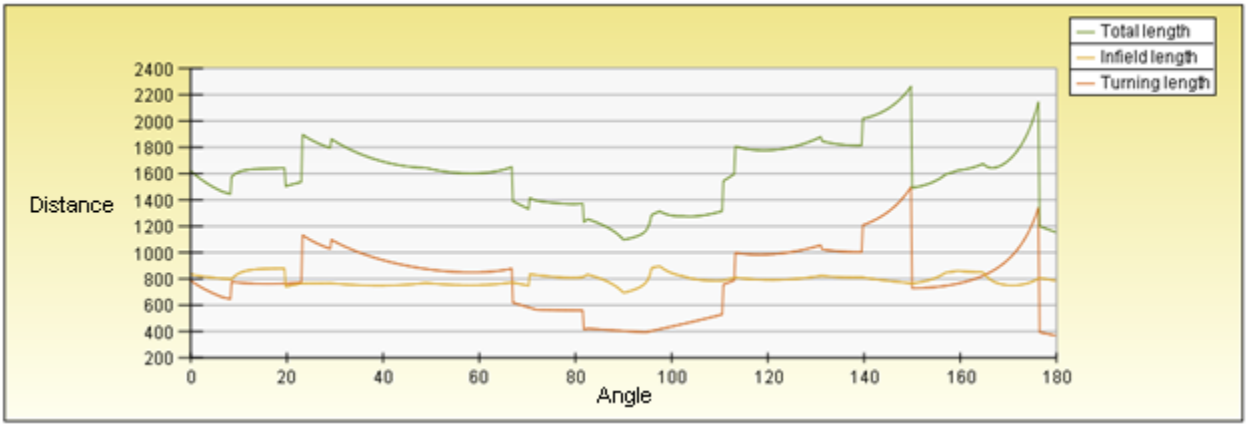


Figure 5. Plot of distance traveled in polygon A when the pass width is 71.

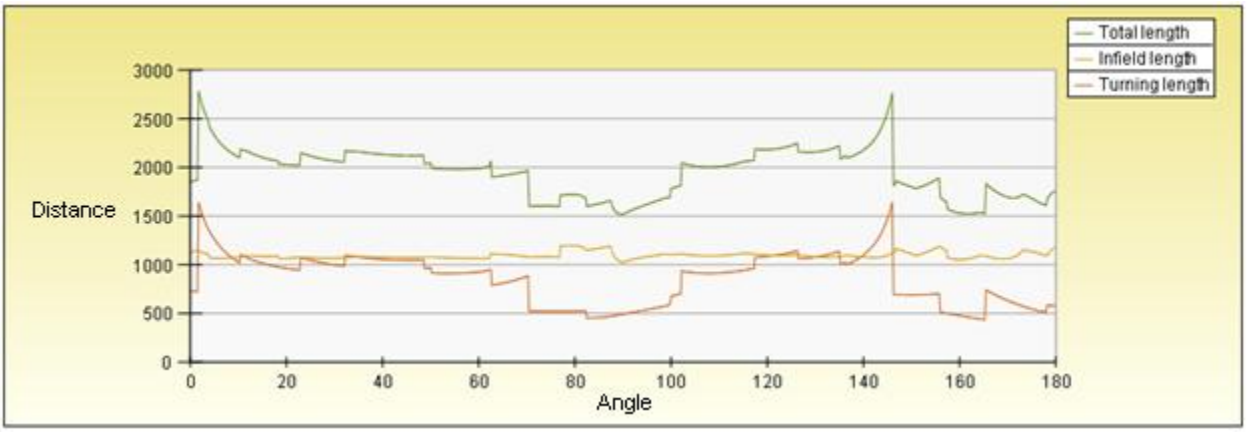


Figure 6. Plot of distance traveled in polygon B when the pass width is 43.

Again, the line representing the total distance traveled generally tracks the behavior of the line representing turning distance only. However, because the turning distance remains relatively the same between angles 80 and 100, the smaller variations in the in-field distance ended up affecting the overall optimal direction that was calculated. Figure 7 depicts what is happening to lead to this result.

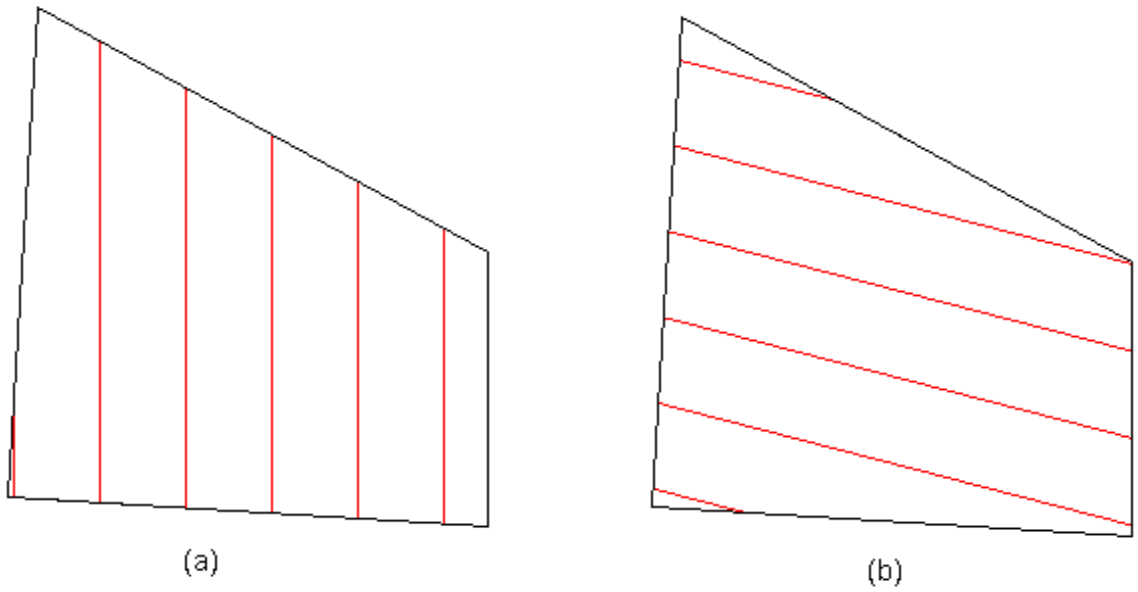


Figure 7. Side by side comparison of passes 43 units apart. (a) Overall optimal angle of 90°. (b) Turning length only optimal angle of 165.25°.

Note that the difference in in-field length is occurring because two of the passes in Figure 7(b) just clip the corners of the polygon, while only one pass in Figure 7(a) does so. However, the dominance of the in-field distance over the turning distance only happens when the pass width is large in comparison to the size of the region. If the region is much larger, as is typically the case with real world fields, the turning length ends up dominating the overall optimal solution. In the typical scenario, the difference in total distance traveled between the two computed optimal directions remained relatively small. Table 1 summarizes the results.

Polygon	Width	Overall optimal angle	Total distance traveled using optimal angle	Number of turns	Turn optimal angle	Total distance traveled using turn optimal angle	Number of turns
A	10	92.25	5816.91	26	91.5	5839.55	26
A	71	90	1097.10	4	179.75	1491.02	4
B	10	85.5	5013.87	23	84.5	5081.34	23
B	43	90	1515.37	6	165.25	1156.44	6

Table 1. Optimal angles and distances traveled

A second thing of note from these preliminary experiments is that the plots of the distance traveled are not smooth curves. This means that there is not a nice, clean, formula whose derivative can be taken to find the true optimal. If the absolute optimal is desired, then costs need to be calculated for all directions between 0° and 180°. However, note that the plots of the total distances traveled and distance traveled in turns generally have peaks and valleys around the critical angles – i.e. the angles of the sides of the polygon and the angles of diagonals of the polygon. For Polygon A, the critical angles are 4.57 degrees, 40.29 degrees, 90.00 degrees, 96.52 degrees, 135.00 degrees, 161.94 and degrees. For Polygon B, the critical angles are 27.51 degrees, 86.57 degrees, 90.00 degrees, 130.33 degrees, 150.95 degrees, 176.42 degrees. To speed up the determination of the "optimal" angle, the search could be limited to areas around those critical angles.

As an aside, another takeaway from these experiments was that as the width was varied, the optimal angle varied drastically, as shown in Figures 8 and 9. In these two figures, the horizontal axis represents pass width and the vertical axis represents optimal angles. This was more an interesting observation instead of one that could be put into use to help restrict the search area.

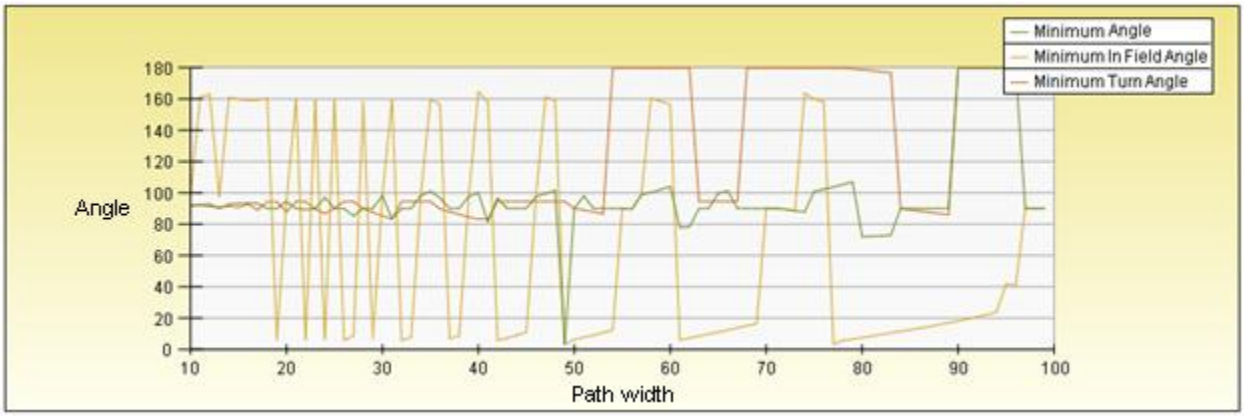


Figure 8. Plot of minimal angles as width is varied from 10 to 100 for polygon A.

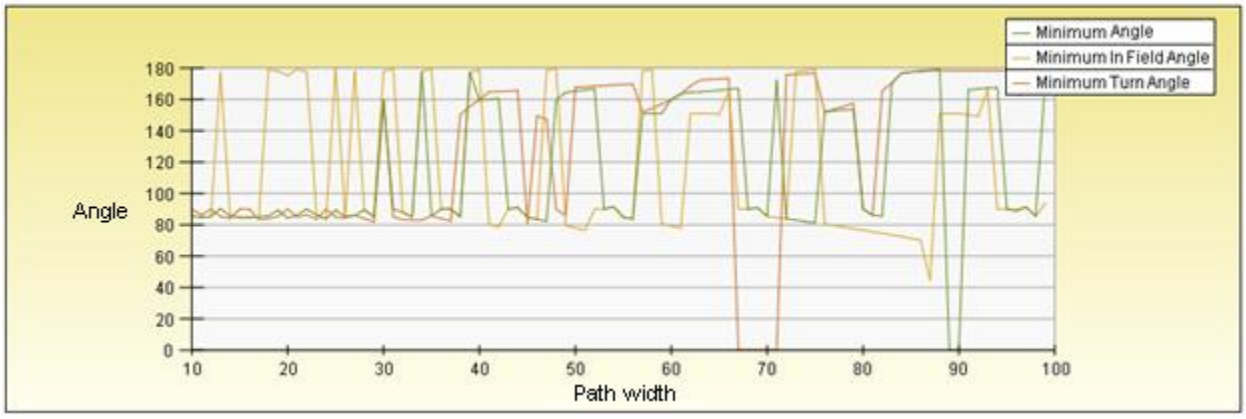


Figure 9. Plot of minimal angles as width is varied from 10 to 100 for polygon B.

Finally, it should be noted that farming vehicles are less efficient in turns. (Jin 2009) Turns are taken at slower speeds than while moving in straight lines. Therefore, from an efficiency standpoint, distance traveled in turns should be weighted heavier than distance traveled in the field. The above experiments were run giving both distances equal weight. Giving the distance traveled in turns more weight, the overall optimal direction more closely tracked the optimal when considering distance traveled in turns. As a result, these preliminary experiments verify the assertion that the in-field distance can be ignored when computing the cost of traveling in a given direction in a field. This does open the possibility of missing the true optimal, but the cost difference between the chosen direction and the optimal direction is relatively small. This in turn allows for more efficient processing of the data since the location and length of each pass did not need to be computed.

2.3 Chosen Cost Function

Based on the findings in section 2.3, a cost formula nearly identical to that used by Jin (2009) was chosen. Ultimately all agricultural cost concerns – minimizing overlap to optimize amount applied, worker productivity, minimizing fuel costs – are significantly affected by distance. Therefore, the cost formula needs to be based on distance traveled. Distance inside the field is the usual straight line distance between two points, but was ignored for reasons discussed

above in section 2.3. Instead only distance traveled in turns was included in the cost formula. Further, like Jin (2007), the distances traveled in straight sections were given less weight than distance traveled while turning.

Where the cost formula differs from that of Jin (2009) is in the type of turns accounted for in the formula. Jin (2009) thoroughly considered the different turning types based on headland size, turning radius and pass width, and set forth the formulas for calculating the respective distance traveled in turns. To simplify calculations in the experiment, it was assumed that all turns are the basic "U-shaped" turn – i.e. that the turning radius equals one half the pass width. However, the algorithm can relatively easily be extended to take into account scenarios where other turning types would be warranted by using the Strategy design pattern. See Gamma et al. (1994).

Formally, the distance traveled while making a single turn on a given edge *i* (DST_i) of the boundary is expressed as:

$$DST_i = 2w / | \tan(|\Theta - \beta_i|) | + \pi r$$

where Θ is the angle of the pass, β_i is the angle of the edge *i* of the field, w is the pass width and r is the turning radius. The basis for this formula can be seen from Figure 10, which depicts the basic "U-shaped" turn scenario. Note that there are three parts to this turn: 1) the distance traveled while moving forward to clear the boundary entirely on both sides of the pass (the part shaded in grey), 2) the distance traveled while moving forward in the headland before making the turn, and 3) the distance traveled in the turn. The distance traveled in the turn, using simple trigonometry, is πr. The distance traveled in the grey area and in the straight away in the headland likewise can be easily calculated using simple trigonometry as $w / | \tan(|\Theta - \beta_i|) |$ each. This yields the above formula.

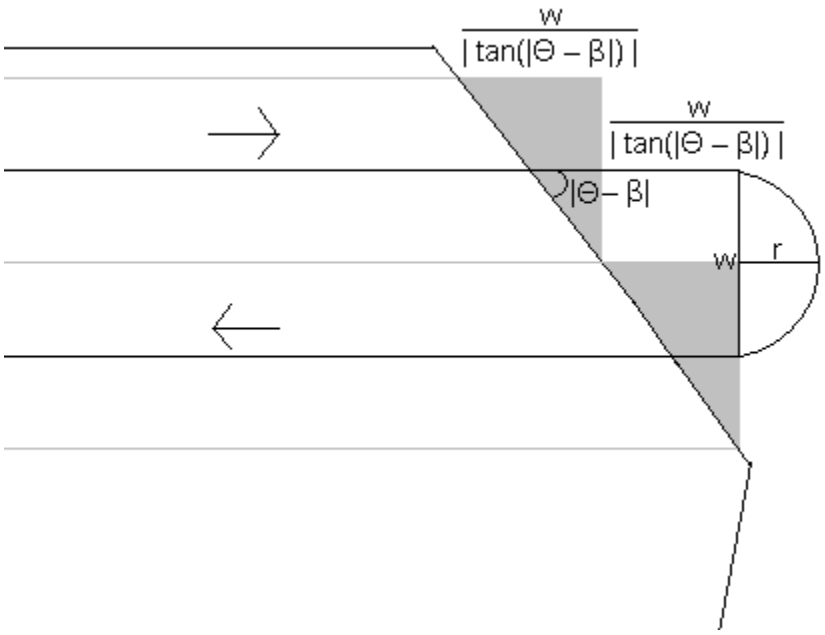


Figure 10. U-Turn

For a given edge, *i*, the number of turns is calculated as:

$$N_i = L_i \sin(|\Theta - \beta_i|) / 2w$$

where, L_i represents the length of edge i. Since the total distance traveled in turns along edge *i* is $DST_i * N_i$, the distance traveled formula for edge *i* can be simplified to:

$$DST_i * N_i$$

$$= (w / | \tan(|\Theta - \beta_i|) | + w / | \tan(|\Theta - \beta_i|) | + \pi r) * (L_i \sin(|\Theta - \beta_i|) / 2w)$$

$$= (w |\cos(|\Theta - \beta_i|)| / \sin(|\Theta - \beta_i|) + w |\cos(|\Theta - \beta_i|)| / \sin(|\Theta - \beta_i|) + \pi w/2) *$$

$$(L_i \sin(|\Theta - \beta_i|) / 2w$$

$$= (L_i * |\cos|\Theta - \beta_i|)| / 2) + (L_i * |\cos(|\Theta - \beta_i|)| / 2) + (\pi L_i * \sin(|\Theta - \beta_i|) /4)$$

The portion of the formula representing the distance moving straight in the headlands was kept separate so as to allow for different weights to be assigned to the three parts of the distance

formula due to efficiency differences. To account for efficiency differences, each of the three parts of the distance formula is multiplied by a cost coefficient, c_k, k = 1…3 which is simply the inverse (time/distance) of the speed in which the vehicle travels during that portion of the turn. Thus modified, the total cost along edge *i* while covering region *j* (TC_{ji}) using parallel paths in the direction Θ_j becomes:

$$TC_{ji} = c_1 (L_i * |\cos| \Theta_j - \beta_i|)| / 2) + c_2 (L_i * |\cos(|\Theta_j - \beta_i|)| / 2) + c_3 (\pi L_i * \sin(|\Theta_j - \beta_i|) /4)$$

The total cost while covering a region in a given direction is simply the sum of the total costs for each of the *n* edges of the field boundary:

$$TC_j = \sum_{i=1\ldots n} TC_{ji}$$

As indicated earlier and discussed in more detail below, the proposed algorithm involves multiple phases consisting of subdividing the field into trapezoidal regions, calculating the optimal direction for each subregion, and merging adjoining regions. The decision whether to merge two adjacent regions, denoted region 1 and region 2, is based on whether merging the two regions yields a cost savings when compared to leaving the regions separate. There are two basic ways in which two regions can be merged: merge while covering each subregion using its own original optimal direction or merge covering the combined regions in the same direction. Each method of merging has its own formula for calculating cost savings. In both cases, if a merge is made, the total cost for the merged region is updated to equal the sum of the original costs of the two regions separately minus the cost savings (CS):

$$TC_1 + TC_2 - CS$$

The first merge scenario, where adjacent subregions are merged while retaining the original directions for each subregion, typically happens where it is possible to smoothly transition a path in one direction to a path in slightly different direction. This in turn happens when the paths intersect the same shared edge, such as in Figure 11. The vertical line in the center represents the shared edge. A merge would be warranted in this scenario if the cost of making the transition from the one region to the other along the original shared edge is less than the cost of making U-turns along the shared edge when covering both regions separately. Thus the cost savings can be expressed as:

$$CS = TC_{1i} + TC_{2i} - CT_i$$

TC_{1i} and TC_{2i} represent the turning cost along shared edge *i* for regions 1 and 2, respectively, in their optimal direction. CT_i represents the transition cost along edge *i* from region 1 to region 2.

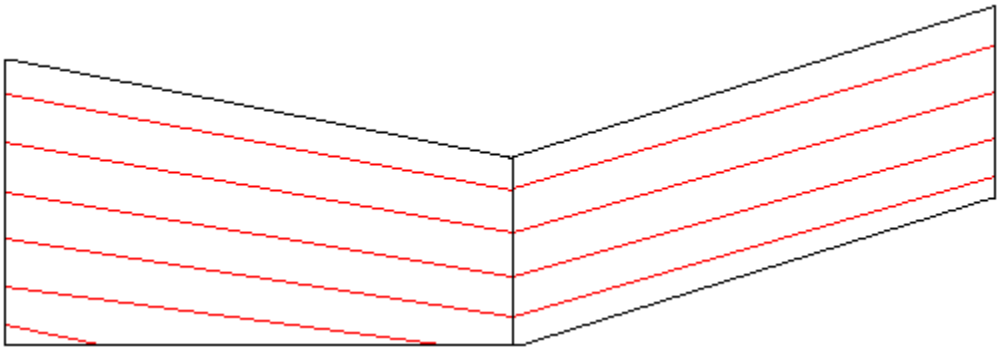


Figure 11. Merging regions while retaining original directions.

In order to formulate the transition cost, it must first be decided how to match up edges from the two subregions, particularly where each region has a different number of turns on the shared edge. There are several approaches for doing so, including leaving the original passes and transitioning from one region to another only when progress in the first region is leading at least one row width ahead of that in the second width, modifying the passes to allow for concentric curves matching on the outside corner, and modifying the passes to allow for concentric curves matching on the inside. The approach used by Jin (2009) was adopted here – namely concentric curves matching on the inside. Figure 12 depicts how this works.

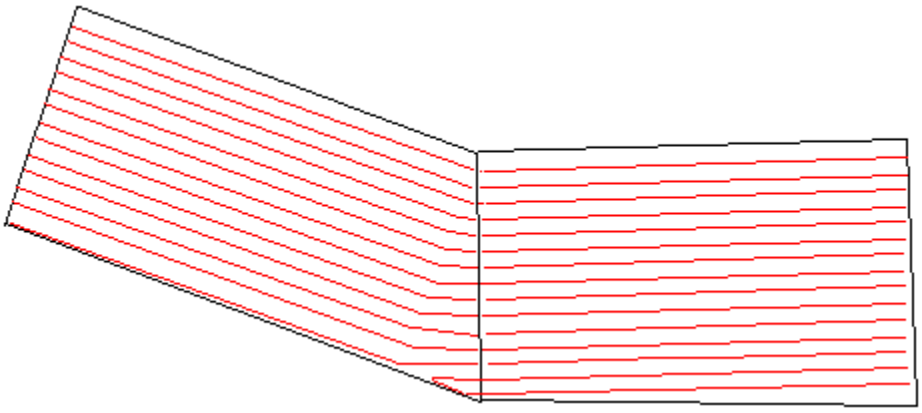


Figure 12. Matching passes between neighboring regions by pairing the swaths from the inside corner. (Jin 2009)

Using this approach the length of the turn during the first transition is simply the arc length of a circle with radius r for an angle difference of $|\Theta_1 - \Theta_2|$: $|\Theta_1 - \Theta_2|$ r. The center of the circle is fixed at the inside corner. So as to maintain a constant pass width of w, for each subsequent transition, the radius increases by w. The number of times this transition is made is simply the minimum of the number of turns on the shared edge for region 1 and the number of turns on the shared edge for region 2:

$$N_i' = \min (L_i \sin(|\Theta_1 - \beta_i|) / 2w, L_i \sin(|\Theta_2 - \beta_i|) / 2w)$$

The total transition cost on edge *i* is therefore

$$CT_i = (|\Theta_1 - \Theta_2|) * N_i' * (r + w * N_i')/2 + C_i$$

Here, C_i is the cost of making u-turns for the remaining passes needed to completely cover the larger of the two regions.

The second merge scenario, where two adjacent regions are joined into a single region which is covered using back and forth passes in the same direction, is depicted in Figure 13. Figure 13(a) shows the optimal directions for the individual regions if covered separately. Figure 13(b) shows the optimal direction for covering the merged region.

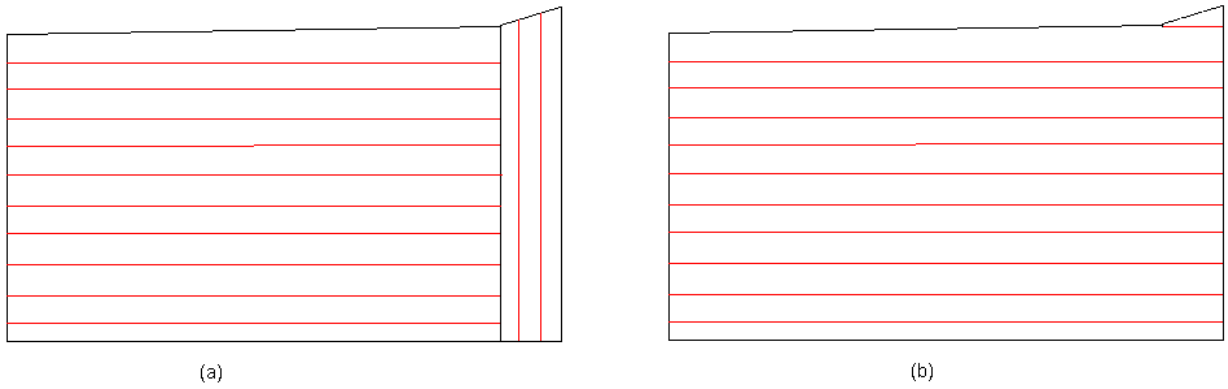


Figure 13. Merging regions using one direction for the entire area. (a) Optimal directions if separately covered. (b) Optimal direction if merged.

When merging two previously unmerged regions, the cost savings in this case is:

$$CS = TC_1 + TC_2 - (TC_{1\Theta} - TC_{i\Theta}) - (TC_{2\Theta} - TC_{i\Theta}).$$

TC_1 and TC_2 represent the total cost for region 1 and 2, respectively, using their own optimal directions. $TC_{1\Theta}$ and $TC_{2\Theta}$ represent the total cost for region 1 and 2, respectively, using passes in direction Θ. $TC_{i\Theta}$ represents the total turning cost along shared edge *i* in direction Θ.

More care is needed when calculating the cost savings of merging a region (A) with a region that was formed by merging two other regions (B and C) while retaining the original directions. See Figure 14 for two options. Denote the left region as A, the middle region as B and the right region as C. Suppose B and C were previously merged with B and C retaining their original directions, as in Figure 14(a). It may be that covering the merged region ABC all in the same direction does not yield a cost savings, but covering AB in the same direction and C in its original direction does yield a cost savings. As a result, the cost savings of merging A with BC must be compared with the cost savings of merging A into B alone.

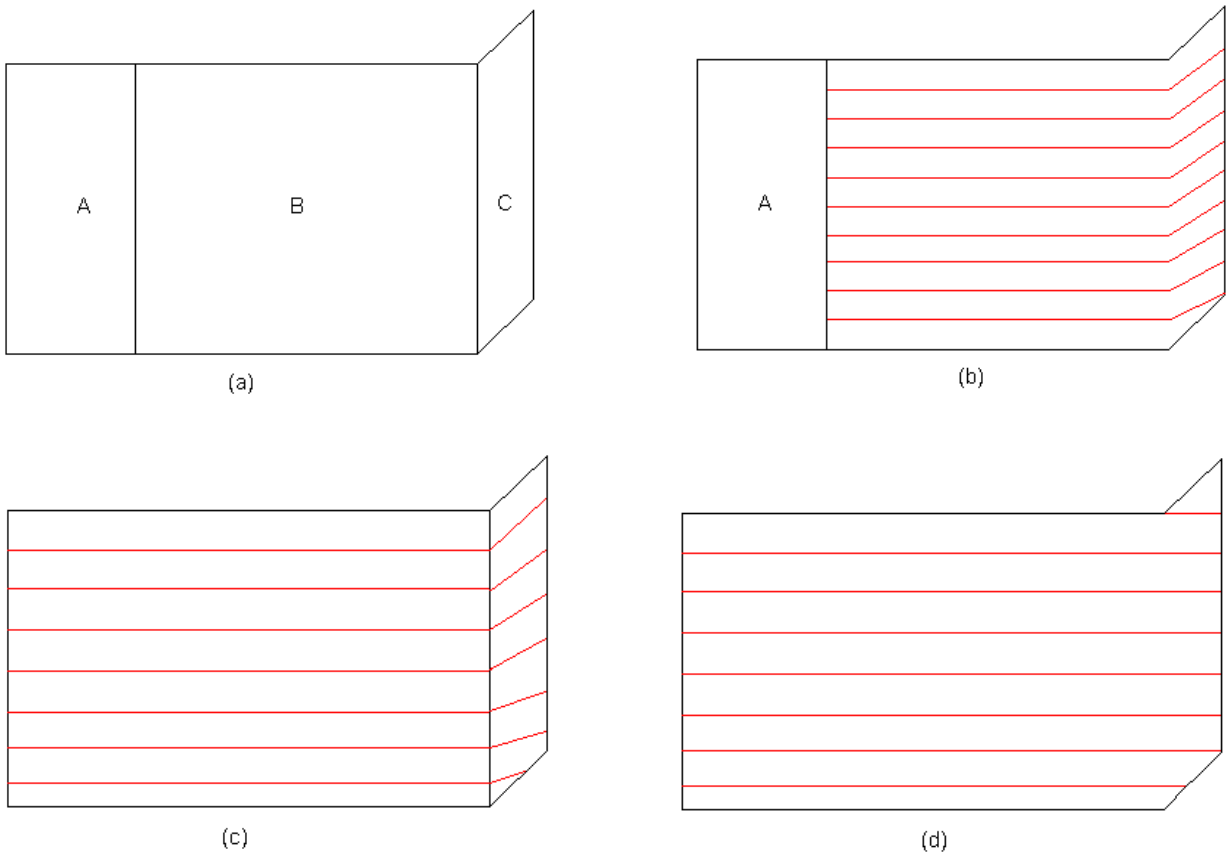


Figure 14. Two options for merging with a previously merged region. (a) The three regions individually. (b) Regions B and C as merged. (c) Option1: merge while covering region C in a different direction from A and B. (d) Option 2: merge while using same direction for all.

CHAPTER 3: ALGORITHM

3.1 Algorithm Overview

The goal of this research is to find an efficient algorithm that is $O(n^2)$ or better and finds a close to optimal solution to the path planning problem. By reduction from the Traveling Salesman Problem, finding the optimal complete coverage solution to a general polygon has been shown to be NP complete, meaning there is no polynomial time algorithm that finds the exact optimal solution. Meuth (2001), Zelinsky (1993). As a result, to achieve a reasonably efficient algorithm, finding the exact optimal solution will be sacrificed.

Two points were noted when formulating a complete coverage algorithm. First, it is relatively easy to calculate the cost of covering the entire field using back and forth passes in a given direction. Iterating from 0° to 180° using a sufficiently small interval would yield a direction that is roughly the optimal direction for doing so. The coverage pattern proposed by any complete coverage path planning algorithm should not have a cost that is greater than the cost of covering the entire field in a single direction. In other words, the sum of the costs of the parts should not exceed the cost of the whole. Further, regardless of the coverage pattern that is proposed, the paths will intersect all of a subset of the edges of the boundary. It stands to reason that if the search for the optimal coverage pattern starts in the same direction as the optimal direction if the field were covered in a single direction using boustrophedon passes, then the resulting solution should be fairly "good". Second, it is much easier to calculate the optimal coverage direction for a rectangle – the optimal being in one of the directions from 0° to 180°, with no subdivision needed. If the field was subdivided into relatively rectangular regions and the optimal calculated for each subregion, the cost savings should be fairly "good." If on top of that all neighboring regions were merged back together if there is a way to more efficiently cover the whole instead of the parts separately, the result would be even closer to the optimal.

Based on the foregoing, as with prior algorithms, our approach involves subdividing the field into subregions where the problem is simpler, and then piecing the parts together, checking at each stage whether it is more efficient to merge or leave the regions separate. To achieve a

better running time, instead of using a recursive divide and conquer approach such as one proposed by Jin (2009) where all possible directions are considered when subdividing the field, a multi-phase plane-sweep algorithm is proposed. The initial sweep direction is chosen to be the normal to the direction in which the field as a whole can be optimally covered using identical back and forth straight passes. From a high level perspective, the algorithm is as follows:

> ***Algorithm****: Quick Optimal Path Planning (QuickOPP)*
>
> ***Input****: A list of n segments, sorted east to west, south to north representing the edges of the field. The edge structure stores the start, end, and possibly additional data such as whether it is passable or impassable. For purposes of this thesis, all sides are assumed to be passable. In the test implementation of this algorithm, the edges of the boundary were stored as part of a doubly connected edge list (DCEL), although other implementations are possible.*
>
> ***Output****: A doubly connected edge list representing the subregions, the coverage path direction for each subregion, and the total coverage cost.*
>
> 1. ***Determine the optimal overall coverage direction****. Determine the direction which attains the minimum turning cost if the entire field was covered in the same direction using boustrophedon paths. Set the sweep direction for the plane-sweep algorithm as the direction normal to the optimal coverage direction. Store the minimum cost.*
> 2. ***Decompose the boundary into trapezoid shaped regions.*** *Using the plane-sweep algorithm, sweep the field in the direction calculated in step 1 to decompose field into (roughly) trapezoidal regions. At each vertex, assuming the width of the new region is larger than the pass width, close the current region and start one or more new regions. The resulting subdivision is represented by a doubly connected edge list (DCEL) and a list of faces representing the interior regions of the field is constructed.*
> 3. ***Calculate optimal coverage for each region.*** *For each interior region that was created in step 2, calculate the direction in which the region can be optimally covered.*
> 4. ***Merge regions.*** *Processing the adjacent regions from right to left, check adjacent regions to determine if it is more cost effective to merge the regions or to leave them separate.*
> 5. ***Return the optimal result.*** *Compare the cost of the proposed solution at the end of step 4 with the cost of covering the entire field in the same direction, and return the solution with the minimum cost.*

The following sections describe in more detail each of the main steps in the algorithm. Example figures are displayed to show the resulting output. Except where otherwise noted, the starting boundary is as shown in Figure 15:

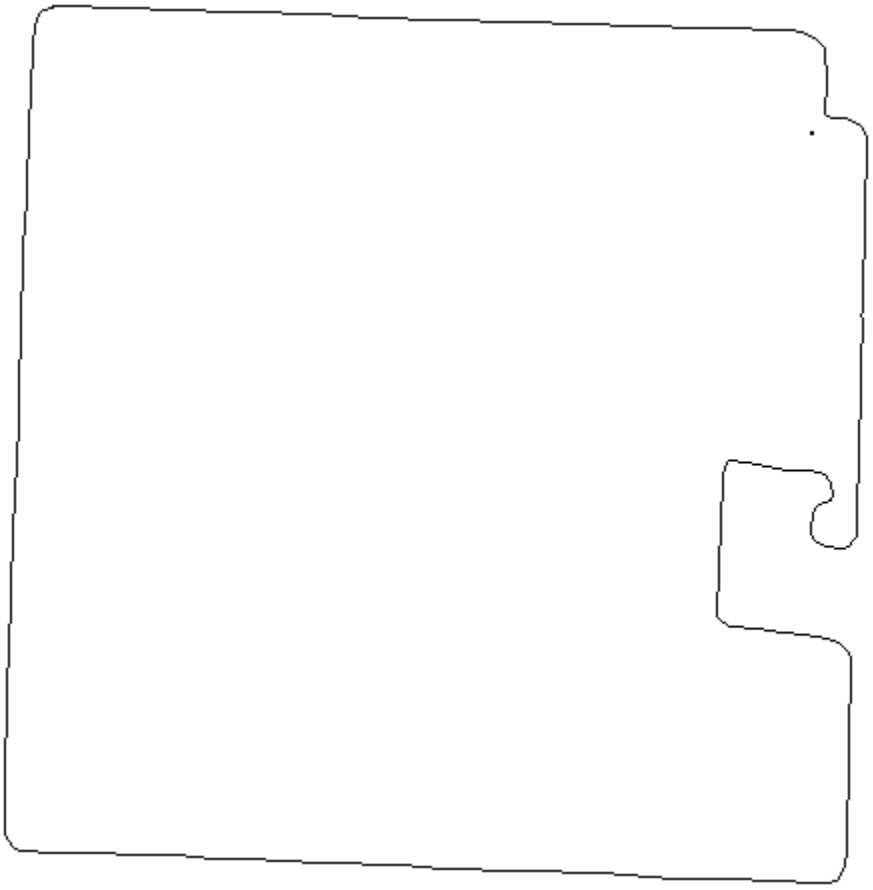


Figure 15. Example Iowa field

3.2 Sweep Direction

The optimality of the paths generated for each subregion is influenced by the sweep direction. Because turns are more expensive than proceeding in a straight line, the optimal paths will tend to follow the longest length of the trapezoid to reduce the number of turns. If the sweep direction is not chosen wisely, the algorithm could fail to consider more optimal directions. To counter act this, the sweeping direction should be one such that after the sweeping algorithm is applied, the resulting subregions will be shaped so as to have optimal coverage directions tending towards the overall optimal direction.

The first step of the algorithm must therefore be to determine in which direction the sweep line in the plane-sweep algorithm should go. As noted earlier, if the entire field was covered in the same direction using boustrophedon passes, one of the directions from 0° to 180° would be the direction it could be optimally covered. Denote this path direction as the initial optimal direction. The algorithm should find a way to cover the field that costs no more than covering the entire field in the initial optimal direction. Further, intuitively, the majority of the field would most likely be optimally covered in the same direction as the initial optimal direction given that regardless of how the field is covered, all passes must intersect a subset of the edges of the field and the initial optimal direction represents the direction that has the least turning cost.

If the field were decomposed so that all long, narrow subregions were long in the same direction as the initial optimal direction, then the optimal paths for each of those subregions would also tend toward that initial optimal direction. The points on the boundary are likely to be closer together than the length of the field. Therefore, if the sweep line was in the initial optimal direction, then the resulting trapezoids would have their longest edges parallel to the sweep line. Since the sweep direction is normal to the sweep line, the problem of determining the direction to perform the plane-sweep algorithm is therefore reduced to calculating the initial optimal path direction for the entire boundary and returning the direction that is normal to that optimal direction. The algorithm for determining the direction to orient is as follows:

***Algorithm**: Compute Sweep Direction*

***Input**: A list of n segments, sorted east to west, south to north representing the edges of the field.*

***Output**: The direction in which to move the sweep line when subdividing the field.*

1. *Call Compute Optimal Direction, passing in the list of n segments.*
2. *Return the normal to the optimal direction.*

***Algorithm**: Compute Optimal Direction*

***Input**: A list of n segments, sorted east to west, south to north representing the edges of the field.*

***Output**: The direction in which the entire region can be optimally covered using parallel straight passes.*

1. *Initialize the overall minimum cost to -1 and the optimal direction to 0*
2. *For each direction from 0° to 180° in the desired increment level*
   a. *Initialize the total cost to 0.*
   b. *For each of the n sides*
      i. *Calculate the turning cost for side i.*
      ii. *Add the turning cost for side i to the total cost.*
   c. *If the total turning cost is less than the overall minimum or if the minimum cost is still in its initial state of -1, update the minimum cost and set the optimal direction to the current direction.*
3. *Return the optimal direction.*

Note that *Compute Optimal Direction* was pulled out into its own algorithm. This is in anticipation of being able to reuse the same algorithm in section 3.4 when the optimal direction for each subregion will need to be computed.

The running time of this phase of the algorithm is easily seen to be O(*dn*) time where *d* is the number of discrete directions between 0° and 180°. Only step 2 takes non-constant time. Step 2 proceeds in a brute force manner, iterating through the directions between 0° and 180°. Obviously, given that there are an infinite number of directions in that range, all possible directions cannot be considered. Rather a discretized subset is considered, computed using a specified increment level. For each direction, the cost of turning on each side of the field is calculated exactly once. Using the cost formulas specified in Chapter 2, that cost calculation can be done in constant time for a given edge. The turning cost needs to be calculated for a total of *n* sides for each of the *d* directions, and so this phase has a running time of O(*dn*). If the number of directions checked in step 2 is held constant, the running time reduces to O(*n*).

Depending on the desired accuracy, the increment size could be increased or decreased. However, care needs to be taken when selecting the increment size as the number of directions checked can dominate the running time. To speed up this algorithm further for regions with a smaller number of edges, use could be made of the observations in Chapter 2 that the optimal direction tends to be the direction in which the turning cost attains a minimum. Thus instead of iterating through all possible directions between 0° and 180°, this phase could first calculate in O(n) time the angles of all the edges and focus on directions around those angles. Since the number of different edge angles is still bounded by 180°, the running time would remain O(n), but the preceding constant would be smaller.

3.3 Trapezoidal Decomposition

The second phase of the algorithm involves subdividing the field into trapezoidal shapes. In this phase, a modified version of the standard plane-sweep algorithm described by Berg et al. (2000) is used to subdivide the field. The field edges are sorted left to right, bottom to top, and are processed in order. For each edge of the field, a vertical edge is created at the left endpoint provided the vertical edge is in the interior of the field. For two edges that have its smallest angle to the left side of the shared endpoint, a vertical edge is created at the right endpoint provided the vertical edge is in the interior of the field. As in the standard plane-sweep algorithm, an active edge list is maintained representing the list of edges, sorted top down, that intersect the sweep line.

Figure 16 depicts the result of a literal interpretation of the plane-sweep algorithm on the sample field.

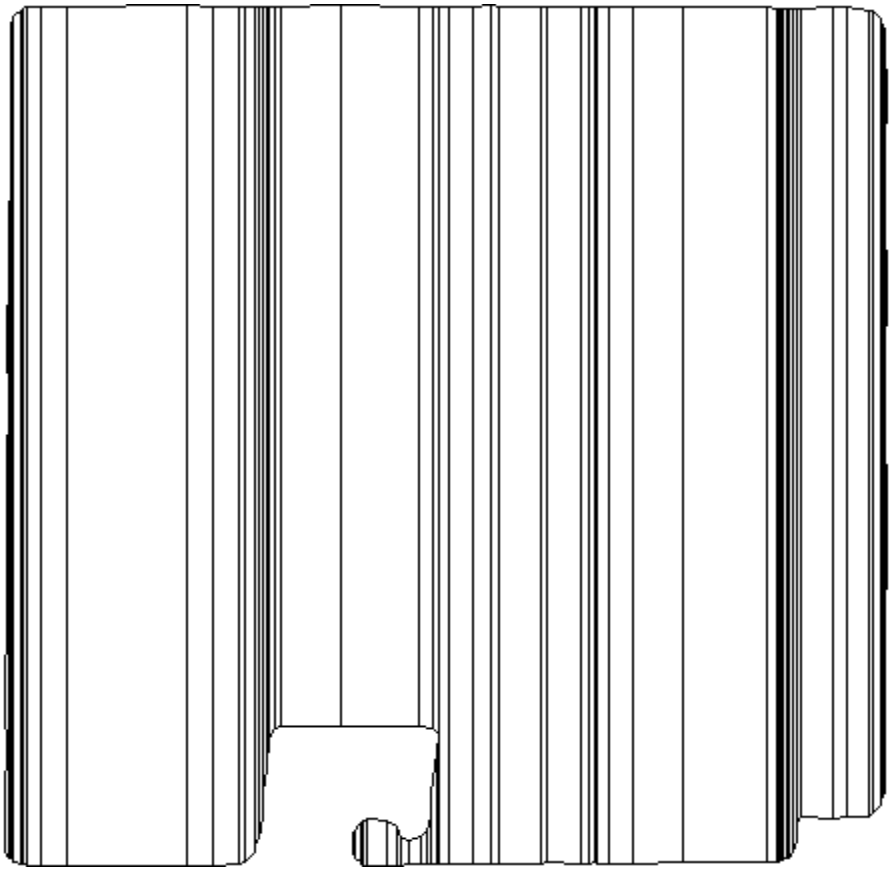


Figure 16. Result of trapezoidal decomposition

As seen from this figure, the vertical lines are highly concentrated in areas where the field boundary curves, particularly along the left and right sides of the boundary. So as to avoid creating regions narrower than the implement, the standard plane-sweep algorithm was modified to keep track of the last vertical line created, and not subdivide the region if the width of the resulting region would be less than the pass (implement) width. Also note from Figure 16 that many of the adjacent regions are rectangular in shape, and would remain rectangular in shape if merged. Oksanen (2007) attempted to avoid this by having a second phase that goes back and merges adjoining regions provided the merged region remained fairly rectangular. However, for better performance, these two phases can be combined into a single decomposition phase by simply not subdividing in the first place if the angle between proposed dividing line and the adjoining edges is approximately 90°. This latter criteria was loosened slightly by avoiding closing a region and starting a new region if the angle between the current edge and the previous edge is approximately 180°. The aim here is to allow for trapezoidal regions generically instead of just rectangular regions. This in turn allows for larger subregions and fewer regions to subsequently merge back together.

There is an exception to the above modifications. Note that when the first point on an interior boundary is encountered, a vertical line created at that point results in one region being terminated and two new regions being started. Similarly, when the last point on an interior boundary is encountered, two regions are terminated and one new region is created. Figure 17 demonstrates these scenarios. Because of the change in connectivity, it is desired that vertical lines always be created at these two critical points.

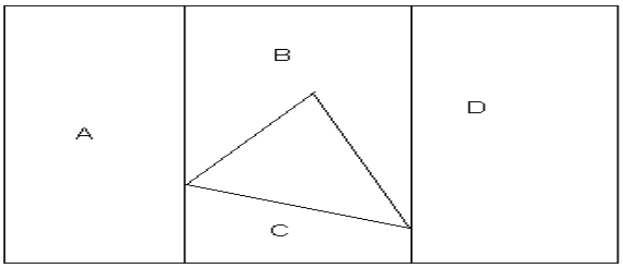


Figure 17. Change in connectivity as vertical edges are created. When the left point on the interior is encountered, polygon A is closed, and polygons B and C are opened. When the right point on the interior is encountered, polygons B and C are closed and polygon D is opened.

The decomposition algorithm is thus:

***Algorithm**: Decompose Into Trapezoids*

***Input**: A list of n segments, sorted east to west, south to north representing the edges of the field, and the direction to rotate the field.*

***Output**: A doubly connected edge list (DCEL) representing the field subdivided into trapezoidal shaped regions.*

1. *Initialize the active edge list to empty.*
2. *Initialize the list of edges to be completed to empty.*
3. *Initialize the location of the last vertical to "not seen"*
4. *For each segment in the sorted input list*
    a. *Mark the edge as "processed".*
    b. *Update the location of the sweep line to the "left" endpoint.*
    c. *If the list of new edges to be completed is not empty and the location of the new edge list falls before the left end point of the current edge, find the index in the active edge list of the edge the vertical comes off of, and connect the vertical to the edges immediately above and below it in the active edge list.*
    d. *Update the active edge list, adding the new edge and removing any edges that are now out of scope.*
    e. *If neither the previous nor the next edge along the boundary has been encountered and the current edge is currently an "upper" edge (i.e, its position in the active edge list is divisible by 2), start an exterior boundary:*
        i. *Get the next segment in the sorted input list. This represents the second edge on the boundary.*
        ii. *Construct a half edge for it, storing the new half edge as the duplicate of the original edge.*
        iii. *Update the active edge list.*
        iv. *Connect the two edges together.*
    f. *Else if neither the previous nor the next edge along the boundary has been processed yet, start an interior boundary:*
        i. *Get the next segment in the sorted input list. This represents the second edge on the boundary.*
        ii. *Mark the second edge as processed.*
        iii. *Update the active edge list.*
        iv. *Connect the two edges together.*
        v. *Close the current region and start two new regions. Construct an edge connecting the left end point of the first edge on the interior boundary to the upper exterior boundary (the edge in the active edge list immediately before the first interior edge). Construct an edge connecting the left end point of the first edge on the interior boundary to the lower exterior boundary (the edge in the active edge list immediately after the second interior edge). The two edges created in this step should form a straight line.*

g. *Else if the current edge is the last edge to be processed on an interior boundary (i.e. both the previous and next edges have already been processed and the current edge isn't the first or last edge in the active edge list)*
    i. *Connect up the edge to the boundary, creating a new edge at the sweep location if necessary. Details of this step are given in step i below.*
    ii. *Close two regions and start a new region. Construct two edges, one from the right end point to the upper exterior boundary, and one from the right end point of the current edge to the lower exterior boundary, connecting the two new edges together. Leave the edges unconnected to the upper and lower exterior boundaries as it is not yet known which edges it would be connected to at the top and bottom. Add the new edges to the list of new edges to be completed.*
h. *Else if the current edge is the last edge to be processed on the exterior boundary*
    i. *Build a new edge at the location of the sweep line, connecting it to the previous edge in the active edge list (the upper exterior boundary), and to the next edge in the active edge list (the lower exterior boundary).*
    ii. *Connect up the edge to the previous or next edge on the boundary, as appropriate, so as to close the exterior boundary.*
i. *Else*
    i. *If the sweep line is more than the pass width from the previous dividing line or if an edge created at the location of the current sweep line would fall outside the field, or if the angle between the current edge and the edge to the left is close to 180°, connect up the edge to the boundary without closing the current region.*
    ii. *Else close the current region and start a new one. Construct a new edge parallel to the sweep line, connecting the left end point of the current edge to the previous edge in the active edge list (if the current edge is a "lower" edge) or to the next edge in the active edge list (if the current edge is an "upper" edge).*

For purposes of the experiments, angles between 170° and 190° were considered "close to 180°"

As thus modified, running this algorithm on the example Iowa field yields the result shown in Figure 18.

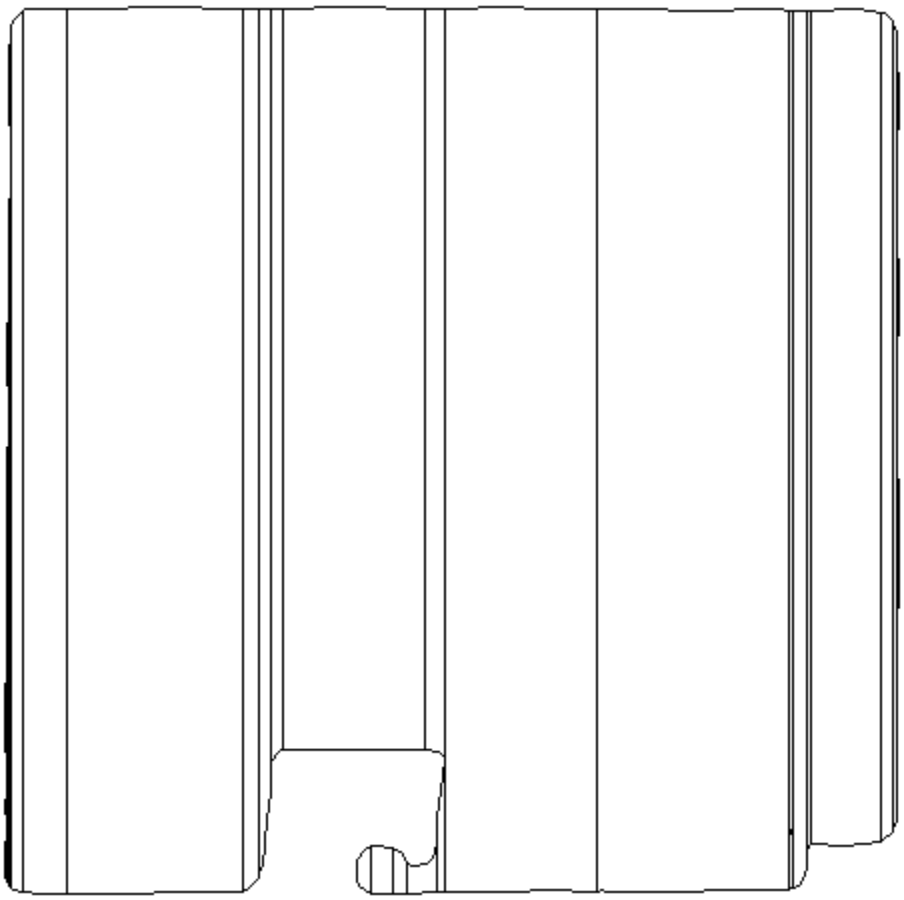


Figure 18. Sample Iowa field after being subdivided.

The running time of this phase takes O(n log(n)) time using a standard plane-sweep algorithm. Each of the *n* sides is processed once. For each side, there is the O(log n) time to insert the edge in the active edge list. Otherwise, deciding whether to close a region and open another region and connecting up the current region takes constant time. The trapezoidal decomposition algorithm results in at most O(n) trapezoids, with at most two new edges created for each vertex of the original boundary. Further, note that each trapezoid has at most two trapezoidal regions adjacent to it on the left and right sides, and no regions adjacent to it on the top or bottom sides. This will be an important fact when calculating the running times of the remaining parts of the overall algorithm.

3.4 Optimal Coverage for Each Region

The third phase of the algorithm is basically a brute force calculation of the optimal coverage direction for the subregion. Considering all angles from 0° to 180° in the desired increment amount (the test algorithm used one degree increments), it calculates the cost of covering the region in that direction. As a result, the algorithm is essentially identical to the algorithm for the first phase, just repeated for each region.

***Algorithm**: Compute Optimal Coverage for All Regions*

***Input**: A list of the subregions of the field. As implemented, this list was in the form of a DCEL as calculated in phase 2.*

***Output**: The direction in which to move the sweep line when subdividing the field.*

1. *For each region in the list of subregions,*
   Call Compute Optimal Direction, passing in the edges associated with the subregion.
   a. *Store the result with the region.*

Figure 19(a) below depicts the results of this phase.

A superficial analysis of this phase of the algorithm would lead to the conclusion that the running time of this phase is $O(n^2)$. The reasoning of such an analysis would proceed as follows: there are O(n) regions as a result of the plane-sweep region, and worst case there are O(n) edges to a region, and to calculate the turning cost would therefore be $O(n^2)$. However, note that an edge can only belong to at most two regions, and so as all of the O(n) regions are processed, each edge is considered at most two times. Thus the running time of this phase is actually O(n).

3.5 Merge of Regions

In the final phase, each pair of adjacent regions are checked to determine whether it is more cost efficient to merge two subregions and cover them using a single pattern, or more cost efficient to keep them separate. Adjoining subregions are processed from right to left. Due to the way in which the overall field was subdivided, each subregion has at least two and at most four adjacent neighbors.

Going into this phase, each subregion is already assigned an optimal direction. When deciding whether to merge two adjoining regions, there are four ways to handle the merge: merge keeping the original optimal directions for each subregion, merge using the optimal direction from one of the two regions for covering the merged region, merge using an entirely new direction, or do not merge. There are also two starting states for the adjoining regions: they either share the same optimal direction or they do not. The following table summarizes these scenarios, and how to handle them.

Scenario	Starting from Same Optimal Direction	Starting from Different Optimal Directions
Merge using existing direction from both	Case A: Easy case – always merge	Case B: Merge if there is a smooth transition and more efficient to make transition instead of leaving separate
Merge using one direction to cover both, that direction being different from direction of optimal for subparts separately	Case C: Can happen, but typically only if the regions separately are long and narrow and together they are broader, in which case the perpendicular direction would be the optimal	Case D: Can happen, particularly if the regions separately are long and narrow and together they are broader.
Merge picking one of the two directions and using that for both	Case E: N/A. This is essentially the same as merge using existing direction from both above.	Case F: Can happen if one region is wide and the other is narrow, and together it makes more sense to use the direction of the wide region.
Don't Merge – Leaving passes as is	Case G: Possible where there is a change in connectivity (Figure 17). Otherwise it would always make more sense to merge.	Case H: Possible

Table 2. Possible scenarios in which two regions can be merged

To handle these scenarios, this phase iterates through each shared edge that divides two adjoining regions. The algorithm recursively calls itself to optimally merge the region on the right side of the shared edge with everything to the right of it. After each recursive call, the algorithm then proceeds to determine whether to merge the two regions abutting the shared edge. When it has been decided whether to merge the two regions or not, control returns to the algorithm, one level up, at which point it proceeds to determine whether not to merge the

preceding region with the region that was just optimally merged. For the typical pair of regions, where there is no change in connectivity, six of the scenarios from Table 2 are considered:

1. Do the two regions have the same optimal direction? If so, compute the cost of covering the two regions in the direction normal to the previously calculated optimal direction and compare it to the cost of covering the two regions in the previously calculated optimal direction. Merge using the most cost effective direction. This step handles cases A and C from Table 2.
2. If the two regions have different optimal directions, determine if the directions intersect the shared edge. If so, calculate the cost savings of merging while covering each region using their own optimal direction, and smoothly transitioning between the regions. Merge if the cost savings is positive. This step handles case B from Table 2.
3. If the two regions have different optimal directions and have not otherwise been merged in step 2, then calculate four cost savings:
    a. the cost savings for covering both regions using the left region's optimal direction,
    b. the cost savings for covering both regions using the right region's optimal direction,
    c. the cost savings for covering both regions using the direction normal to the left region's optimal direction,
    d. and the cost savings for covering both regions using the direction normal to the right region's optimal direction.

    Compare the cost savings, using the direction corresponding to the greatest cost savings. If all cost savings are negative, then do not merge. This step handles cases D, F and H from Table 2
4. If the left and right regions are merged based on the foregoing, and the right region is a merged region of smaller regions, a final check is made to see if it would be more cost effective to only merge the left region with the left part of the right merged region, effectively splitting the merge. See Figure 14 in section 2.3.

In the case where the shared edge represents the location where one region ends and two regions starts, the algorithm first calculates, using the above three step process, the cost savings

of merging the region on the left with the region on the lower right. It then computes the cost savings of merging the region on the left with the region on the upper right instead. The region on the left is then ultimately merged with whichever region on the right has the greatest cost savings. The algorithm handles the case where the shared edge represents the location where two regions end and one region starts similarly. It first calculates the cost savings of merging the lower left region with the region on the right, and compares that cost savings with the cost savings of merging the upper left region with the region on the right. The region on the right is then ultimately merged with whichever region on the left has the greatest cost savings. If no merge can be done with a positive cost savings, no merge is performed.

Figure 19 shows the before and after of this phase of the algorithm on areas where there is a change in connectivity. Note that even though all optimal paths are in the same direction, only the upper left and lower right triangle is merged with the middle rectangle. Due to the edges of the field, it is not possible to process both the top and bottom regions in a single up and down, left to right pass, and so a decision was made as to which regions to merge.

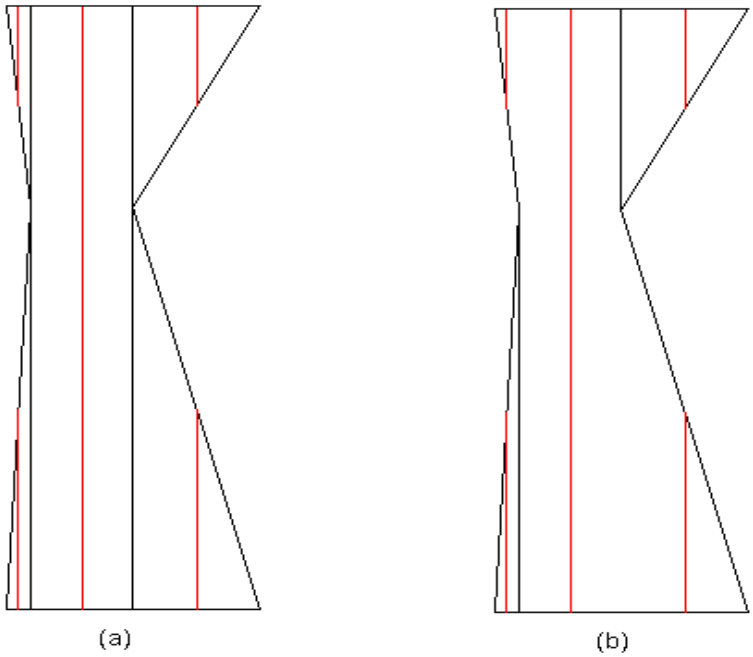


Figure 19. Sample output of the merging phase. (a) The field just after phase 3 and before the merging. (b) The field at the end of phase 4 after merging.

Figure 20 shows the sample Iowa field previously depicted in Figure 18 after several rounds of merging. Note how near the lower left corner there are two black vertical lines. These lines represent that small portion below the small interior boundary that was not merged. Like the upper right region in Figure 19, this area was not merged with either the region to its right or the region to its left due to it being more cost effective to merge those regions with the larger region above the interior boundary.

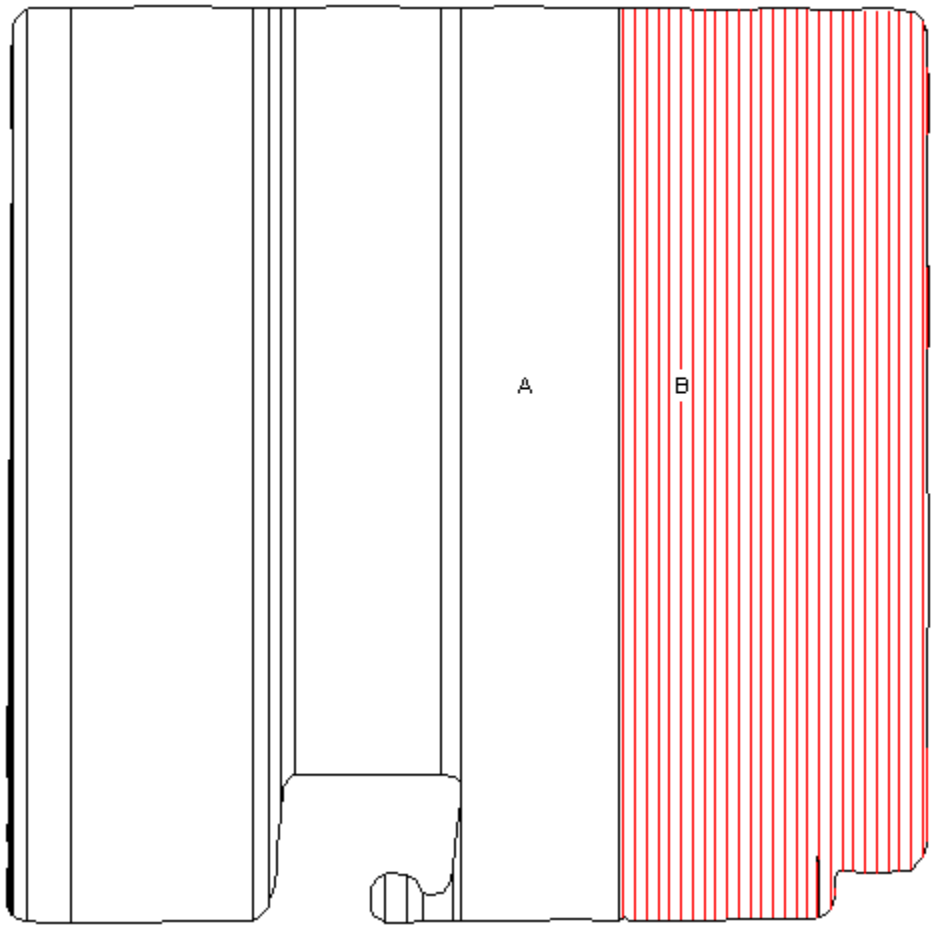


Figure 20. Sample Iowa field after several rounds of merging, just before merging region A with merged region B.

The specific algorithms used during the merge phase are as follows:

***Algorithm**: Merge*

***Input**: A DCEL representing the field subdivided into trapezoids. Stored with each face in the DCEL is the merged face it is currently part of. The merged face is initialized to NULL.*

***Output**: A DCEL representing the field "optimally" covered.*

1. *Set firstFace equal to the left most face in the DCEL.*
2. *Call MergeBestChild(firstFace, NULL)*
3. *Return the updated DCEL.*

***Algorithm:** Merge Best Adjacent Region*

***Input**: The subregion to be processed (regionToBeProcessed) and the subregion just visited (cameFromRegion). Stored with each region is the merged region it is currently part of.*

***Output**: The right portion of the field, starting from the subregion to be processed, with all subregions "optimally" merged.*

1. *Set currentEdge equal to any edge on the subregion to be processed. The current edge represents the common edge between two faces.*
2. *Mark regionToBeProcessed as visited.*
3. *Set startEdge equal to current edge.*
4. *While currentEdge is not equal to the start edge*
    a. *Set secondRegion equal to the face of currentEdge's twin.*
    b. *If secondRegion is not equal to cameFromRegion and it adjoins on the right,*
        i. *Recursively call Merge Best Adjacent Region(secondRegion, regionToBeProcessed)*
        ii. *Call Merge Two Regions(regionToBeProcessed, secondFace, currentEdge).*
    c. *Else if secondRegion is equal to the cameFromRegion and it adjoins regionToBeProcessed on the left, add it to the list of left edges to process later.*
5. *For each edge, leftEdge, in the list of left edges to process later*
    a. *If the region on the opposite side of the edge is not NULL and it was not already visited*
        i. *Recursively call Merge Best Adjacent Region(the opposing region, regionToBeProcessed);*
        ii. *Call Merge Two Regions(regionToBeProcessed, leftEdge);*
6. *Return the subregion that was just processed (i.e., regionToBeProcessed)*

***Algorithm**: Merge Two Regions*

***Input**: Two subregions and the shared edge between the regions. Stored with each region is the merged region that it is currently part of. The second region is already assumed to be optimally merged for all regions to the right of it.*

***Output**: The regions merged or not, depending on whether it was cost effective to do so.*

1. *Set previousMergedFace equal to the merged region currently assigned to the first region*
2. *If the two regions have the same optimal direction. In this scenario, the regions will be merged.*
    a. *Calculate the cost savings if both regions were covered in the direction perpendicular to the individual optimal direction.*
    b. *If the cost savings is greater than 0, the regions are merged using the perpendicular direction as the new optimal. Store with the merged region the current cost savings for merging.*
    c. *Otherwise the regions are merged using the original optimal direction and store with the merged region the current cost savings – which is the amount saved by not needing to turn along the common edge.*
3. *Otherwise*
    a. *If the optimal paths covering the two regions intersect the shared edge, calculate the cost savings for merging the two edges. Merge if the cost savings is greater than 0.*
    b. *If the optimal paths do not intersect the shared edge,*
        i. *Calculate the cost savings (using the formula from step 1), for covering the two regions using the optimal direction for region 1.*
        ii. *Calculate the cost savings (using the formula from step 1), for covering the two regions using the optimal direction for region 2.*
        iii. *Calculate the cost savings (using the formula from step 1), for covering the two regions using the direction perpendicular to the optimal direction for region 1.*
        iv. *Calculate the cost savings (using the formula from step 1), for covering the two regions using the direction perpendicular to the optimal direction for region 2.*
        v. *Determine which of these four options yields the largest cost savings.*
        vi. *If the largest cost savings is greater than 0, merge using the corresponding coverage pattern and store with the merged region the current cost savings.*
4. *If the regions were merged and the first region was previously associated with another merged region*
    a. *Compare the savings for using the current merged region with the previous merged region.*
    b. *If it is more cost effective to merge the first region with the second region instead of with the regions that it was previously merged with, remove the first region from the merged region it was previously associated with and set its merged region to the region created in step 3.*
5. *If the regions were merged and the second region was previously associated with another merged region*
    a. *Compare the savings for using the current merged region with the previous merged region.*

b. If it is more cost effective to merge the second region with the first region instead of with the regions that it was previously merged with, remove the second region from the merged region it was previously associated with and set its merged region to the region created in step 3.

Of all the phases of the overall algorithm, this phase has the longest running time. Each trapezoidal region is processed in order, comparing it with at most two adjoining regions to its right and two adjoining regions to its left. As previously mentioned in section 3.3, the trapezoidal decomposition yields O(n) shared edges, meaning there are O(n) pairs of regions to consider. For each pair of regions, the cost of merging the region in each of six different ways are calculated and compared to the cost of covering the regions separately. In the case of merging a region with a previously merged region, one more cost calculation is made to determine if it would be cost effective to split the previously merged region. Finally, two additional costs are calculated and compared in the cases where there is a change of connectivity. The calculation of these costs takes O(n) time as the costs must be calculated for each side and then summed. In the worst case, there are O(n) sides in the right region since that region could represent a merger of one or more other regions. As a result, the overall running time of the fourth phase is $O(n^2)$. Since the running time of all other phases of this algorithm is less than the running time of this phase, the overall running time of the Quick Optimal Path Planning algorithm is also $O(n^2)$.

3.6 Order of Traversal of Remaining Regions

Now that there is a plan in place for covering each region of the field, and the bounding edges of each region, the next step would be to determine the order in which each subregion should be processed to minimize the transition time. Although not implemented in the test program, this step can be implemented by viewing each region as a node in a binary graph and traversing the graph in a depth-first manner. Choset (1997, 2000) Further, if we let the distance between two adjacent regions be 1 unit, and the distance between two non-adjacent regions be 1 plus the distance between the ending point of the final pass through the first region and the starting point of the first pass through the second region the distance between the center points of each region, a best-first walk of the graph could be performed instead to determine the order in

which to process the regions. One such best-first search algorithm that could be used is the A* search algorithm.

3.7 Correctness of the Algorithm

In order to achieve reasonable performance of the complete coverage path planning algorithm, finding the precise optimal way for covering the entire field has willingly been sacrificed. Nevertheless, it can be shown that the above algorithm finds a reasonable, relatively "optimal" solution.

As discussed in section 3.2 above, the algorithm first computes the initial optimal direction for covering the field in a single direction using parallel paths. The initial optimal direction is computed in a brute force manner, iterating through each angle between 0° and 180° at the desired increment level, picking the direction with the minimum distance traveled as the initial optimal direction. The total distance traveled for the actual overall optimal solution must be less than or equal to the distance traveled covering the field in the initial optimal direction. During the third phase of the algorithm, the optimal direction for each trapezoidal (convex) subregion is computed, also in a brute force manner, again with the optimal direction for a subregion being the direction with the minimum distance traveled. See Figure 21 for how the sample Iowa field looks after the third phase of the algorithm. As with the initial optimal direction, the sum of the distances traveled in each subregion while covering it using its own optimal direction is also an upper bound on the actual overall optimal solution. The main idea of the algorithm is once the optimal direction for each subregion has been determined, to try to incrementally determine if this upper bound can be improved upon by merging adjacent regions.

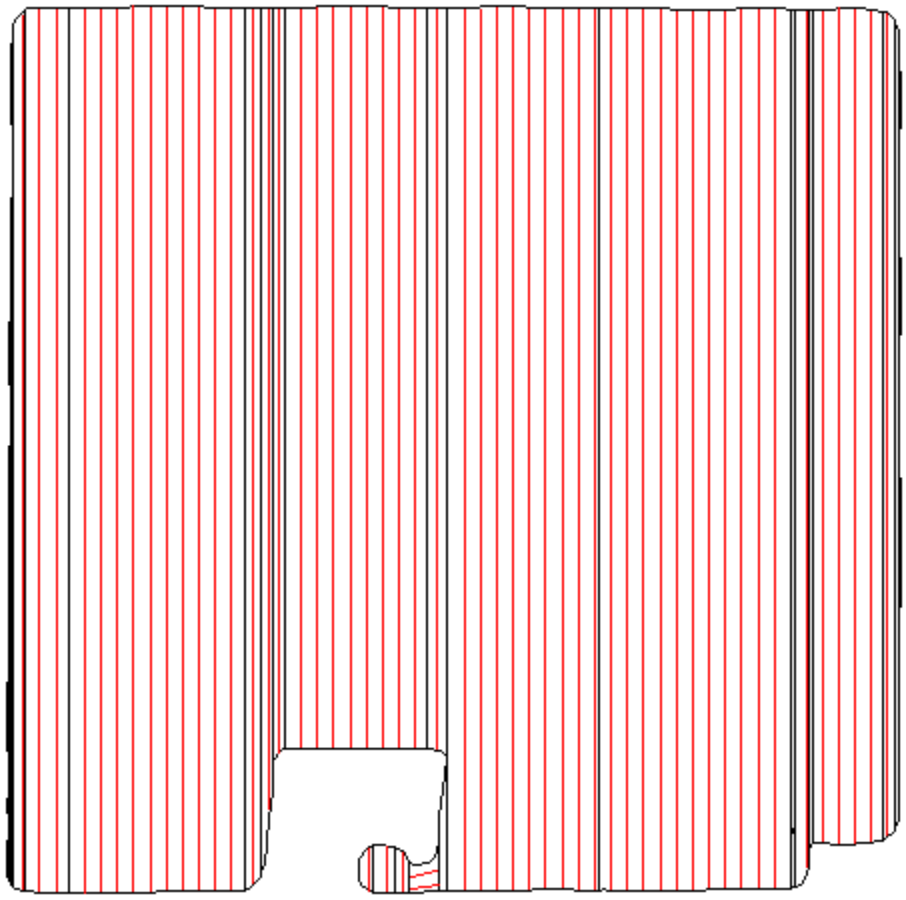


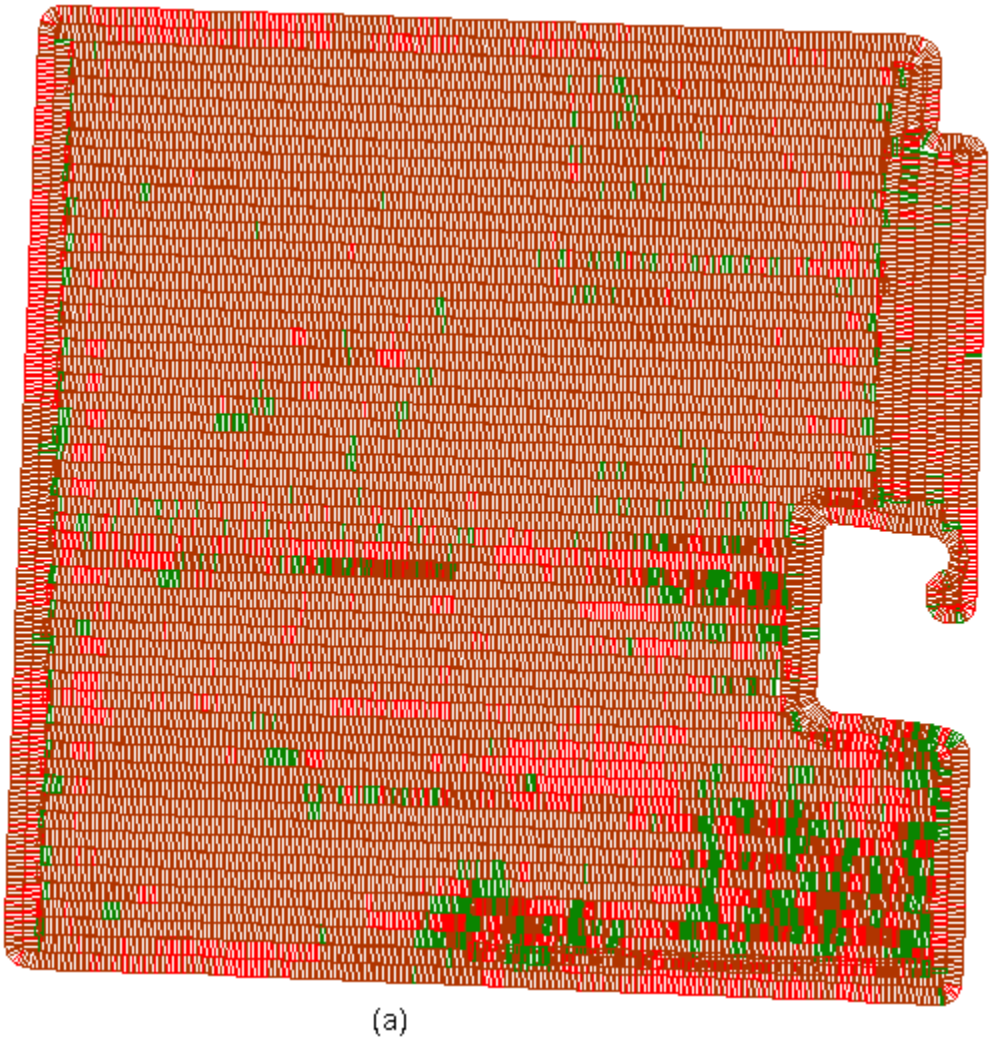
Figure 21. Sample Iowa field with optimal directions for each sub region after phase 3.

During the merge phase of the algorithm, at the end of each call to *Merge Best Adjacent Region*, the subregion being processed is relatively optimally merged (or not merged) with all regions to the right. As a result, at the end of each call to *Merge Best Adjacent Region*, the total distance traveled using the solution so far is no greater than the distance traveled if each subregion were covered separately. To see this, consider how *Merge Best Adjacent Region* operates. It first recursively calls itself passing in the adjacent region to the right as the "region to be processed" and the current region being processed as the "came from region". This does not stop until the algorithm reaches the rightmost region. In the rightmost region, everything to the right of that (which is nothing) is trivially optimally merged. Now assume that at the end of the i'th call to *Merge Best Adjacent Region*, region being processed is relatively optimally merged or

not with everything to the right. On return from the i'th call to *Merge Best Adjacent Region*, what was the region being processed is now the adjacent region to the region being processed. Then *Merge Regions* gets called with the region being processed as the first region, the adjacent region as the second region, and the shared edge. *Merge Regions* then determines whether two can be merged in a manner that saves on cost (i.e. reduces the distance traveled). If so, *Merge Regions* returns with the merged face. If not, it is more optimal to leave the two regions separate. Either way, after each call to *Merge Regions*, the region being processed and the adjacent region are relatively optimally merged (or not merged), and so *Merge Best Adjacent Region* in turn returns with the region being processed optimally merged (or not merged) with everything to the right. See Figure 22(b) for the results of the merging phase on the sample Iowa field. Therefore when the merging phase is completed, the distance traveled using the proposed solution is better than or equal to the distance traveled covering each region separately. The overall algorithm then compares the distance traveled using this solution with the distance traveled using the initial optimal direction, and returns the solution with the minimum distance traveled. In this manner, the solution reached is no greater than either the initial optimal solution or the solution of covering each region separately.

CHAPTER 4: EXPERIMENTS AND RESULTS

The QuickOPP algorithm was coded in C++ using Microsoft Visual Studio 2008. All tests were run on a laptop with a 2.80 GHz Intel Core 2(™) Dual CPU, processor with 2.96 Gb of RAM. Real field boundaries were created from data logged by the John Deere GreenStar 2 System™, and unloaded into GreenStar Apex desktop software (Apex). Images of the as-applied or as-processed field are taken from data unloaded into Apex.



(a)

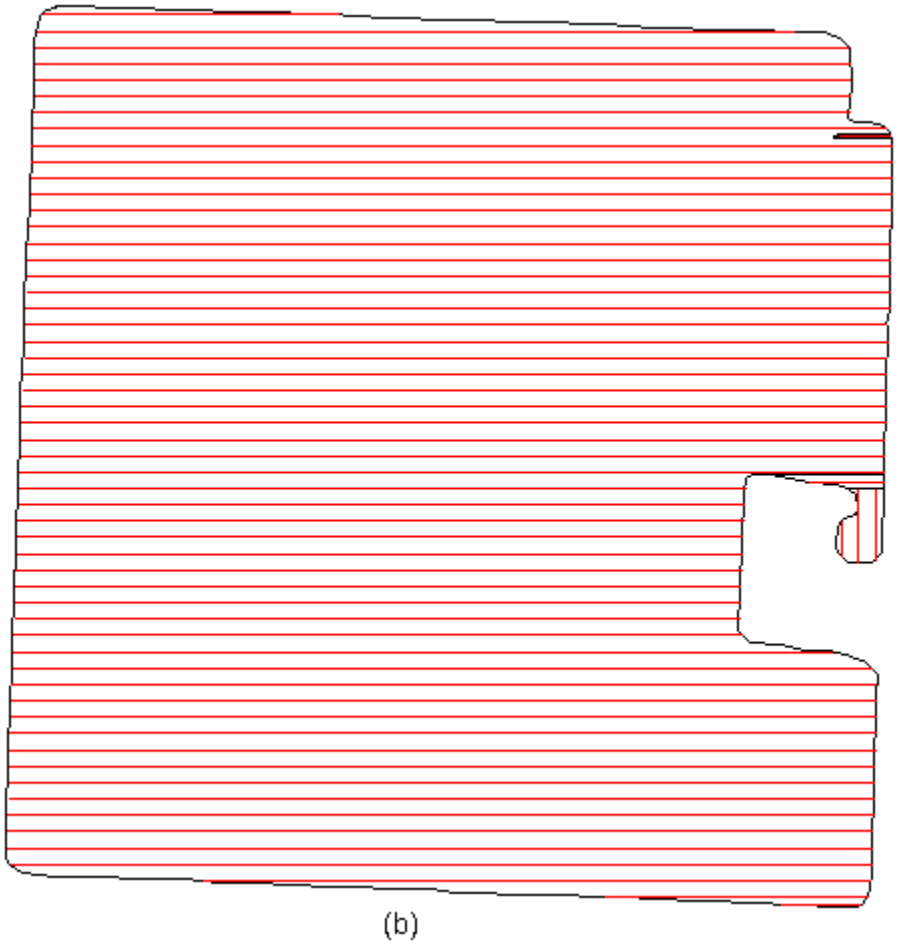
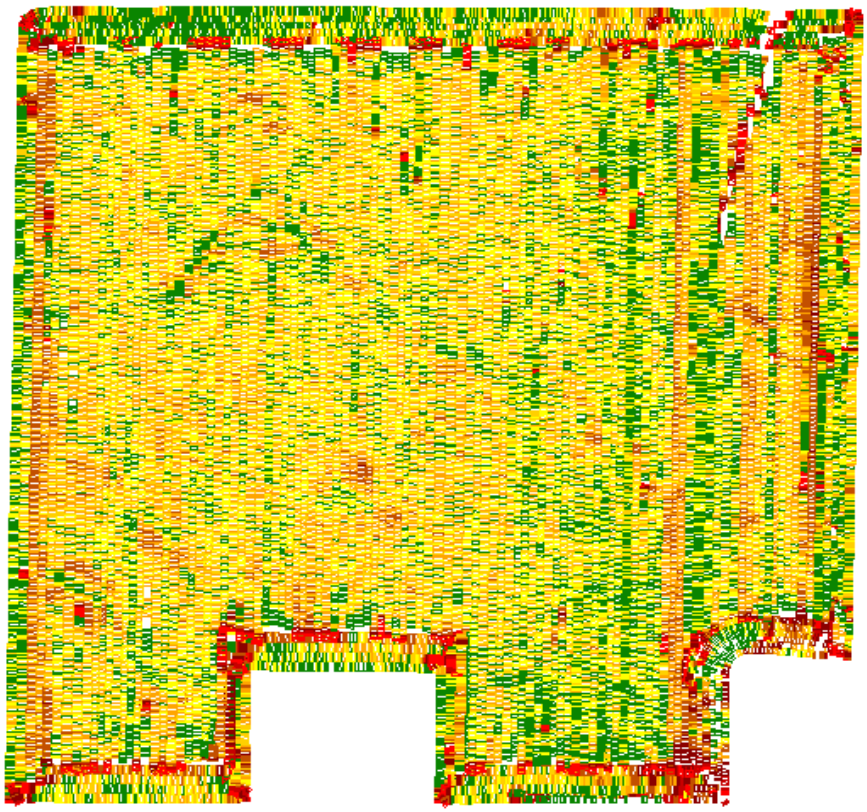
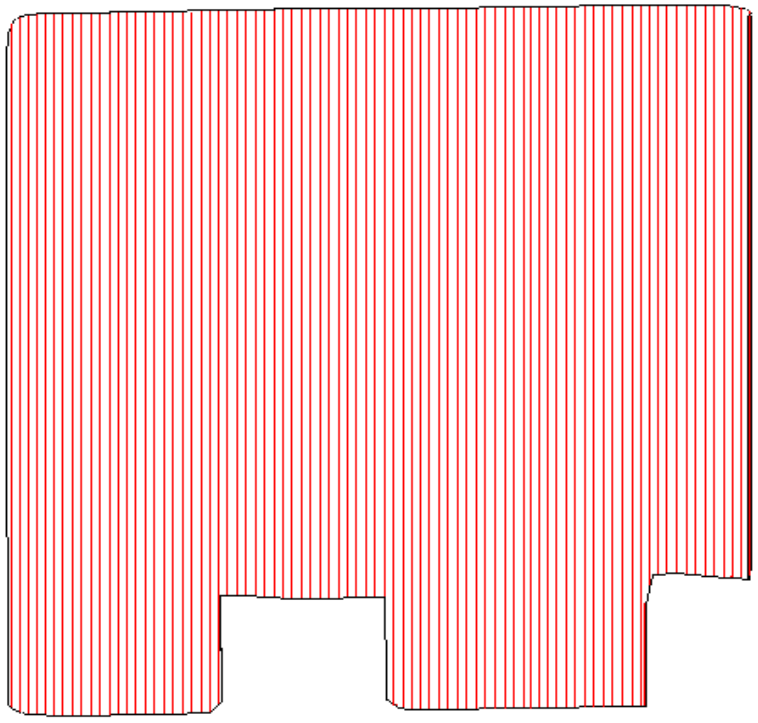


Figure 22. Iowa field 1. (a) As fertilized by the farmer. (b) As proposed by QuickOPP.

The QuickOPP algorithm was first tested against one of the sample fields provided with the Apex software. Although it is a "sample" field in Apex, the field is an actual Iowa field located in central Iowa. The boundary was auto-generated in Apex from application data logged in 2007. Figure 22(a) shows the route the farmer took to fertilize the field. Notice that the farmer first drove the entire boundary a couple times and made a few more vertical passes along the right side before covering the rest of the field using horizontal passes. The pass width was 46'40", or approximately 15 meters. Figure 22(b) shows the route proposed by QuickOPP using the same pass width. Notice that the route proposed by QuickOPP is generally the same as the "as driven" route, except it eliminates five turns by eliminating the vertical passes along the right and extending the existing horizontal passes.



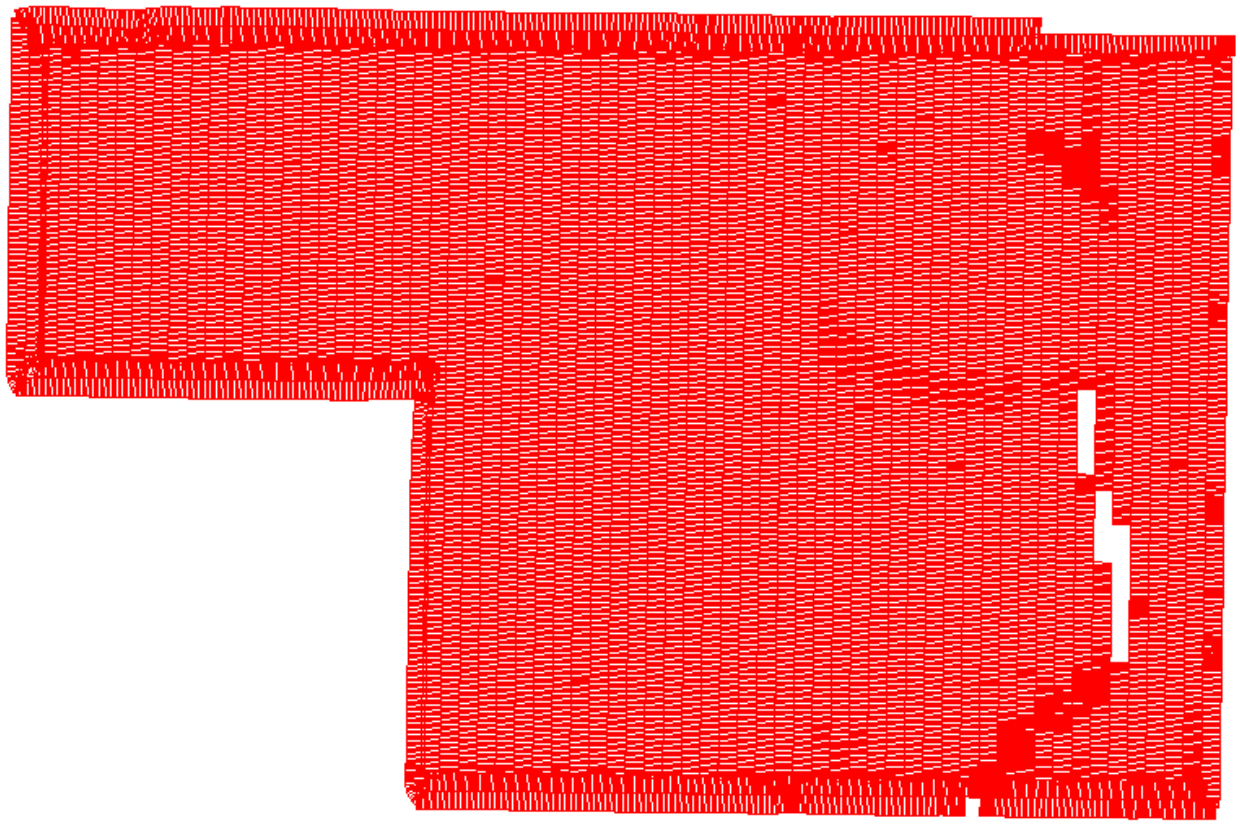
(a)



(b)

Figure 23. Iowa field 2 (a) As harvested by the farmer. (b) As proposed by QuickOPP.

The second field that QuickOPP was tested against is also an actual field from Iowa included in Apex's sample dataset. This time, the boundary was auto-generated in Apex from harvest data logged in the field back in 2006. Figure 23(a) shows the directions traveled while the farmer was harvesting the field. The width of the depicted passes was approximately 20 meters. As with the previous Iowa field, the farmer here first drove the boundaries a couple times before processing the interior. Figure 23(b) depicts the route proposed by QuickOPP. The general direction proposed by QuickOPP is identical to the "as driven" route, not including the driving of the headlands.



(a)

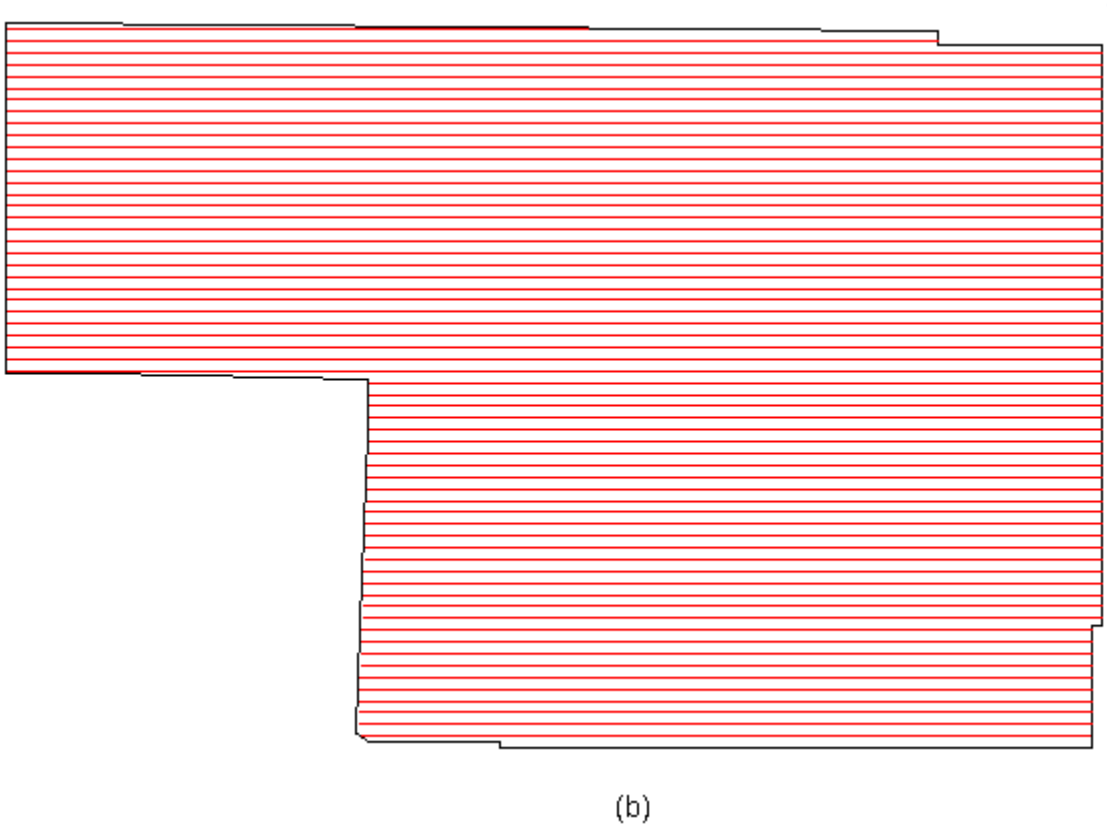


Figure 24. Iowa field 3 (a) As planted by the farmer. (b) As proposed by QuickOPP

The third field the algorithm was tested with was another field from Iowa included in Apex's sample dataset. This time, the boundary is from data points logged in the field while driving the field boundaries. Figure 24(a) shows the directions traveled while the farmer was planting corn. The blank spots in the middle of the field are from where there was a loss of GPS signal, and do not indicate interior boundaries. The pass width was approximately 20 meters. The farmer here chose to drive the northern and southern boundaries in a horizontal manner, and the remaining of the field in a vertical manner. Figure 24(b) depicts the route proposed by QuickOPP. QuickOPP recommends that the field be covered using horizontal straight passes instead of vertical. The QuickOPP result has a turning cost of 9667.69 meters, while as driven by the farmer, the turning cost is 10170.70 meters. This eliminates a significant number of turns and results in a calculated savings of 503.01 meters.

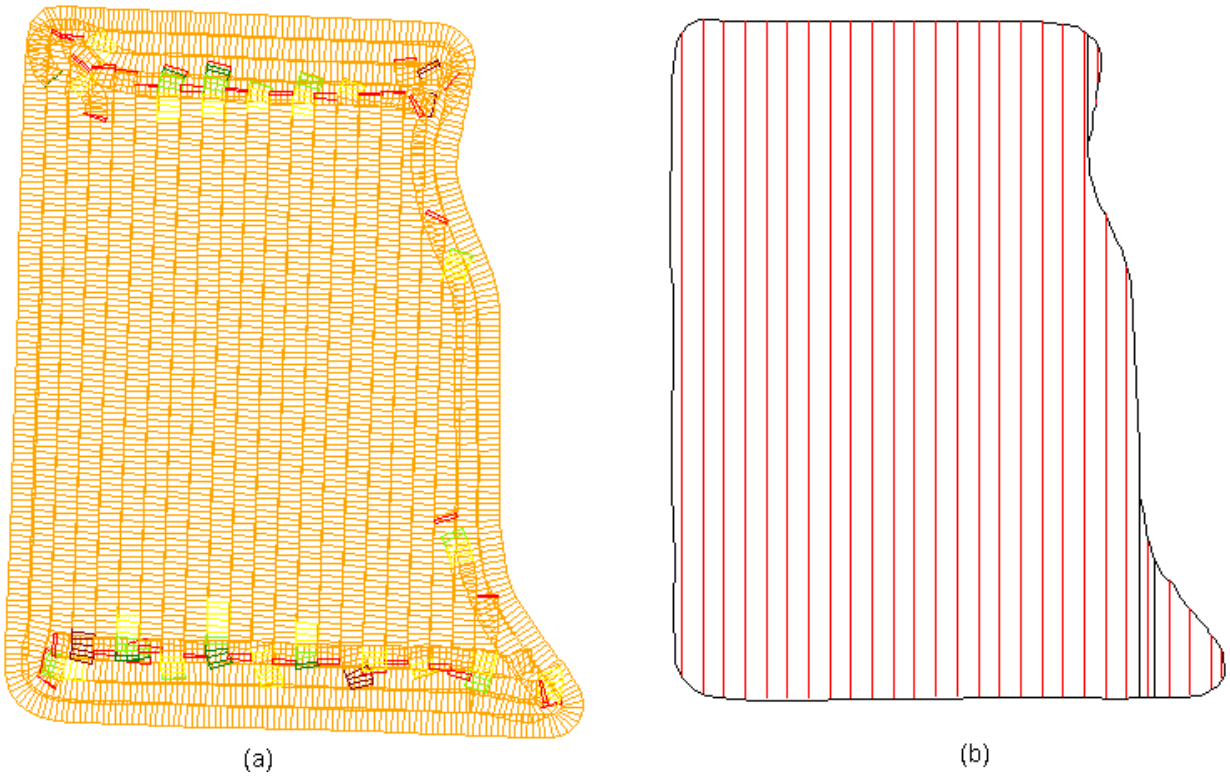


Figure 25. North Dakota field 4 (a) As fertilized by the farmer. (b) As proposed by QuickOPP

Figure 25 compares the results of QuickOPP with how a farmer fertilized a field in North Dakota. As before, Figure 25(a) shows how the farmer drove the field. In this instance, the pass width was approximately 13 meters. Note that the farmer drove the boundary along the north, east and south sides, and that no headland was used along the west edge of the field. Figure 25(b) shows the path QuickOPP would have taken. The results are fairly equivalent as due to the turnings needed on the north, east and south sides, headlands would be required anyway.

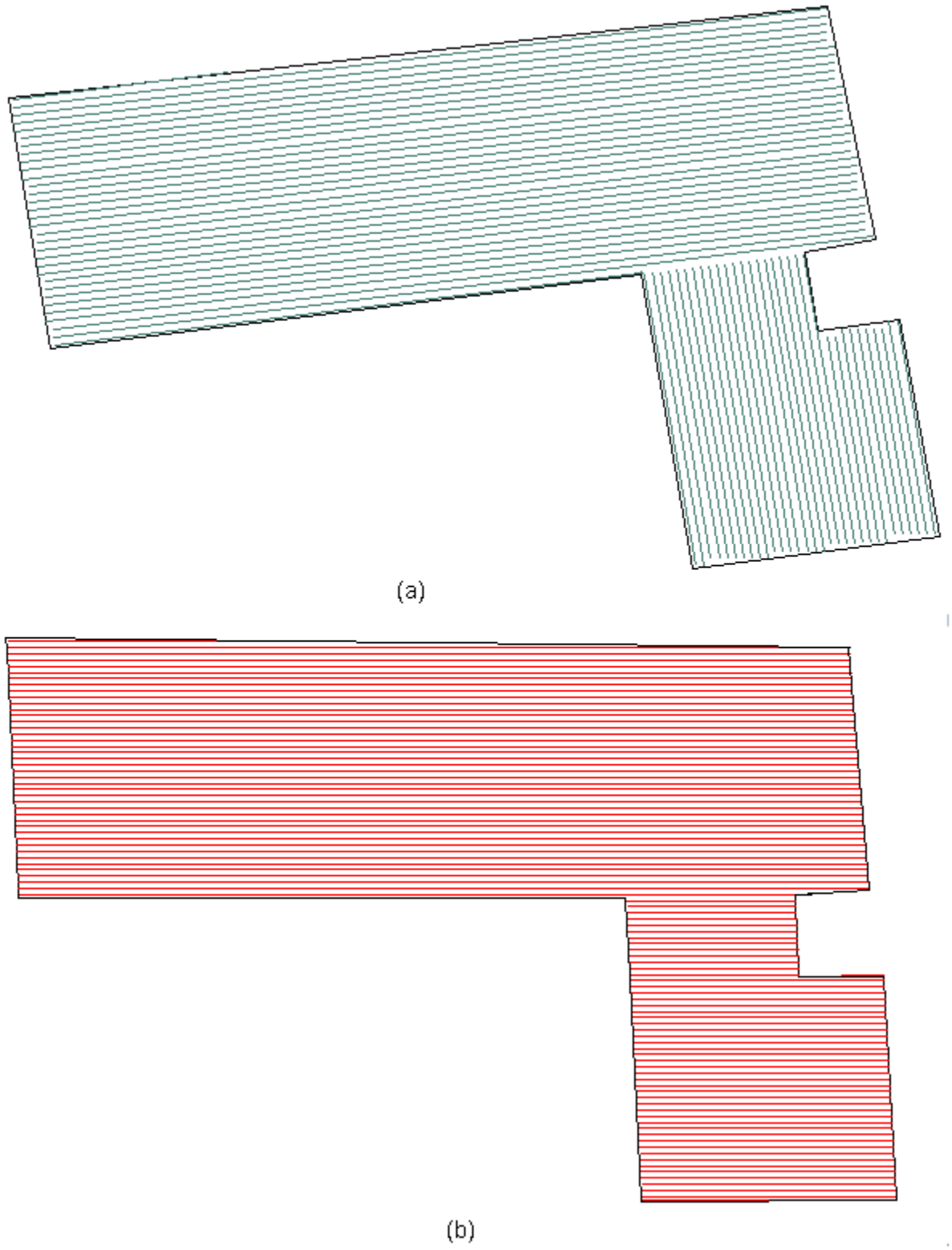


Figure 26. Ohio field 1. (a) Jin's approach with pass width of 30 feet. (b) QuickOPP result.

QuickOPP was also tested against some of the same fields Jin (2009) used to test his algorithm (OPP) against. The application created by Jin (2009) was also available for side by side performance comparisons. Figure 26 compares the results of QuickOPP and OPP. Both algorithms ran in under a second. At first glance it would seem that the solution proposed by Jin's algorithm is more optimal. However, during the merge phase, QuickOPP did consider if it were more cost efficient to use the approach in Figure 26(a), yet concluded that the approach in Figure 26(b) was actually better assuming all turns are U-turns. The total turning cost as calculated by QuickOPP using Jin's proposed path is 4549.59 feet assuming all turns are U-turns. The total turning cost using the paths recommended by QuickOPP is 4342.88 feet, a savings of about 206.71 feet. Consider the closed "handle" portion alone, including a new edge dividing it from the rest of the field. Although it is cheapest to cover that region in isolation using vertical passes, when taken in context of the entire field, doing so imposes turning costs on that new edge – costs that were avoided by covering the region horizontally. It turns out the savings of not having to turn on this new edge when covered horizontally outweighs the savings when covering the region vertically. This result would be different if different types of turns were taken into account, or if the three parts of the cost formula discussed in Chapter 2 were weighted differently. In fact, if more weight is given to the turning cost portion of the formula, QuickOPP does return with the same solution as that proposed by OPP.

Figure 27 is a comparison of the results of OPP and QuickOPP using a second field from Ohio. In both cases, the pass width was set to 30 feet. Note that the recommended path directions are identical in both approaches. However, the QuickOPP algorithm only took approximately 3 seconds to reach this result in comparison, while OPP took approximately 16 minutes before the final solution was reached. This is significant cost savings considering this dataset only contains 32 points with three interior boundaries. Depending on the local geography, actual fields can have significantly more points and interior boundaries. For example, in parts of North Dakota there are lots of small lakes. Figure 28 depicts four fields located in that state. These fields are from a dataset containing twenty-three (23) farms with several fields each, and are typical of the size and shapes of the fields in that dataset.

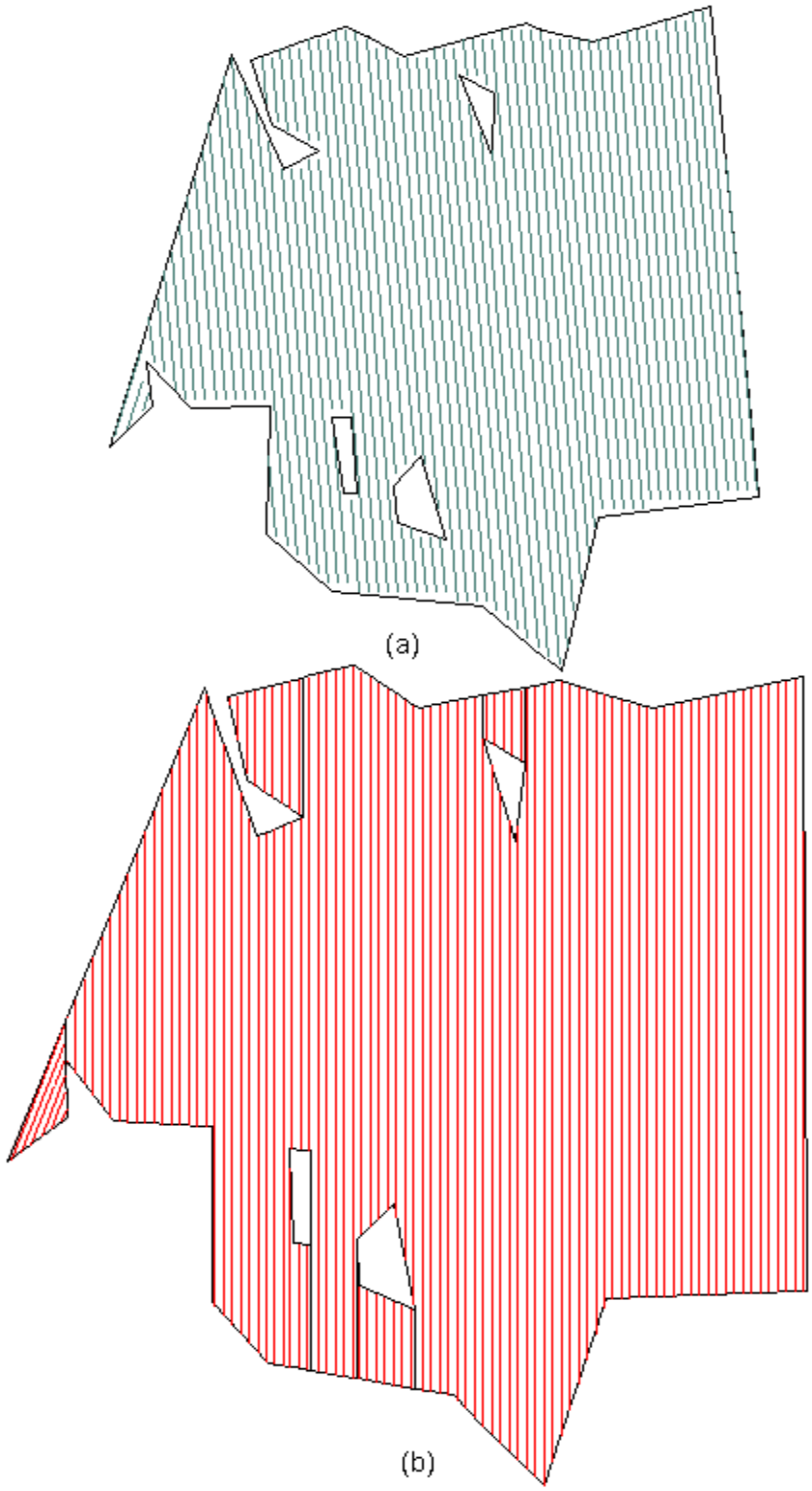


Figure 27. Ohio field 2: (a) Jin's approach with pass width of 30 feet (16 min to process) (b) QuickOPP result (3234 milliseconds to process)

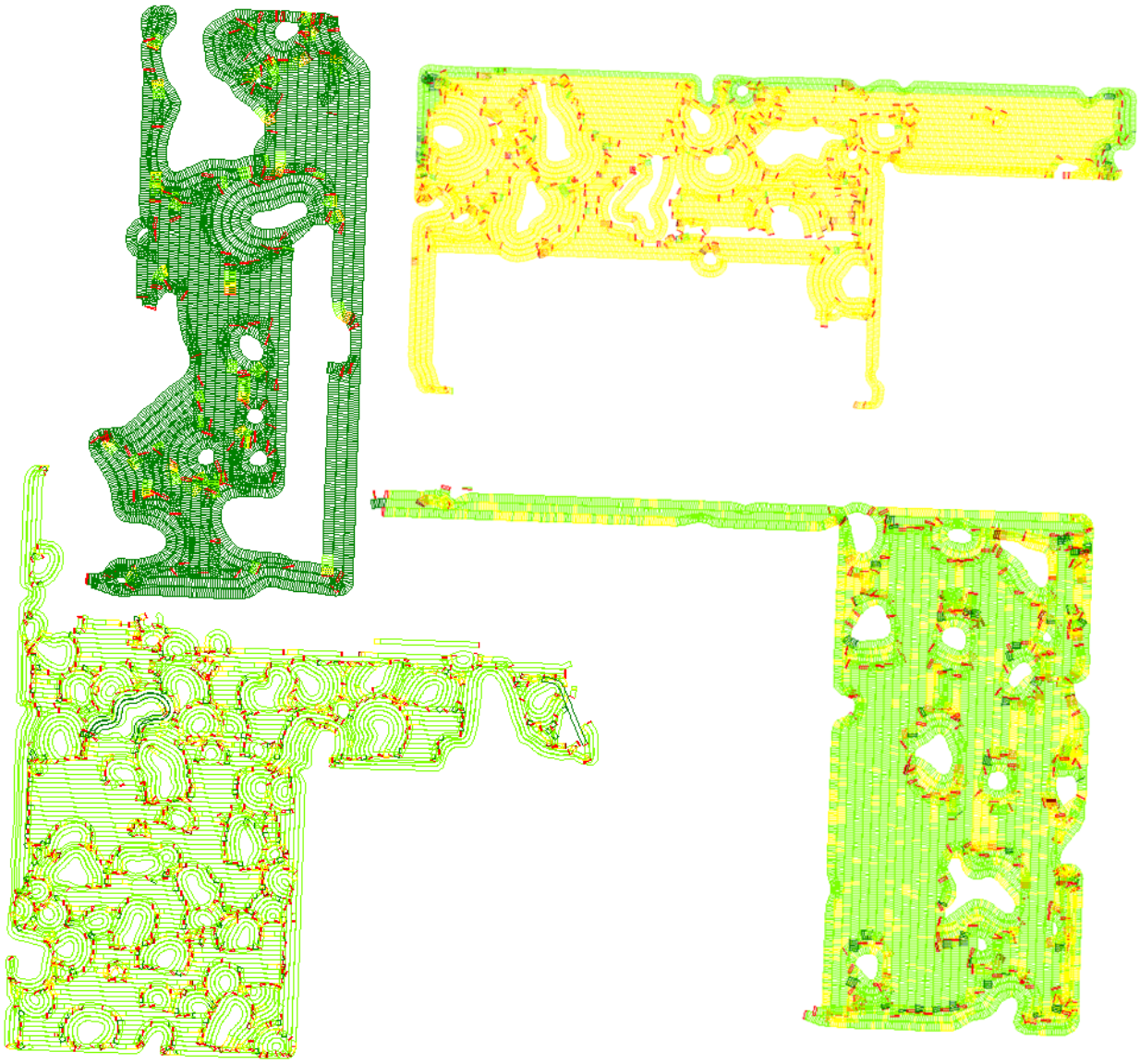


Figure 28 Four fields from North Dakota with a large number of interior boundaries and data points

Finally, Figure 29 compares the results of this algorithm with that of Jin's using the example from Fabre et al. (2001). The results were identical with a pass width of 30 feet, and slightly different with a pass width of 20 feet. There are a few things worth noting here. First, even though QuickOPP doesn't yet account for the differing turning types, it came up with

identical results as Jin's OPP algorithm for a pass width of 30 feet. Using U-turns at the field boundaries, the OPP is the more optimal approach for a pass width of 20 feet but the distance traveled for the QuickOPP solution is not significantly more. Second, if the sweeping algorithm swept in the normal direction, the merging algorithm yields the same results as Figure 29(b). To consider more options for subdividing the field, the algorithm could be run twice, once to sweep in the overall optimal direction and once to sweep in the normal direction. Third, note that the QuickOPP approach did not merge the two sides of the field into one. The reason is that it assumed that all boundaries were impassible, and so a transition would need to be made to the lower right corner of the right half so as to be able to completely cover the right half using parallel straight paths. Finally, it turns out that neither OPP's nor QuickOPP's solution is the optimal solution in the case of a 20 foot pass width. Through trial and error, it was discovered that a yet more optimal solution is as presented in Figure 30. It is similar to the approach shown in Figure 29(b), but the passes are parallel to the northeast instead of the southwest sides of the upper right section. Nevertheless, the distance traveled using QuickOPP's solution is only 61 feet more than the distance traveled using the more optimal solution, having a turning cost of 2403.72 as compared to a turning cost of 2342.83 using the more optimal solution. Table 3 summarizes the results of the above experiments.

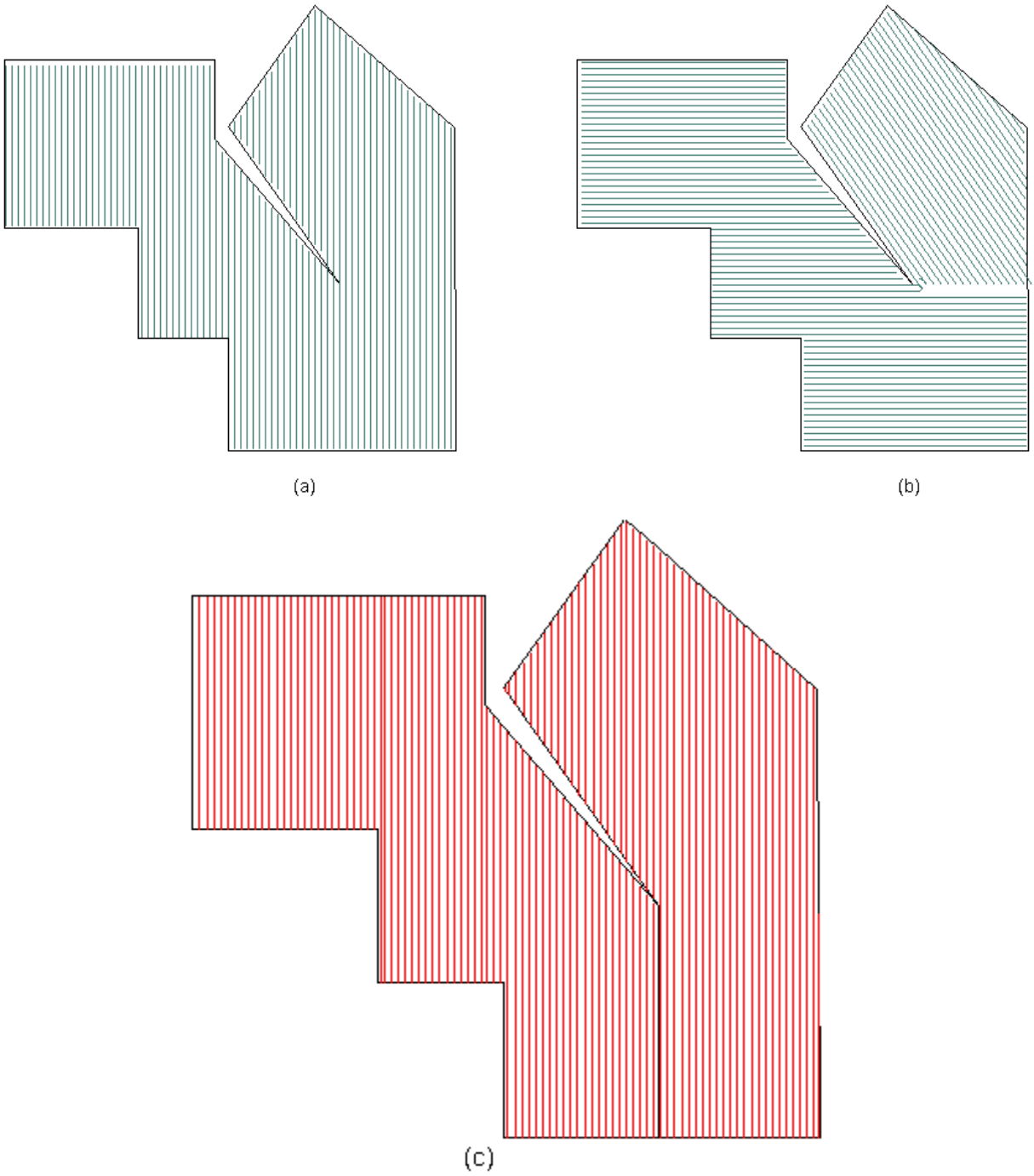


Figure 29. Fabre et al's field example. (a) Jin's results with pass width of 30 feet. (b) Jin's results with pass width of 20 feet. (c) QuickOPP results for both pass width of 30 feet and 20 feet.

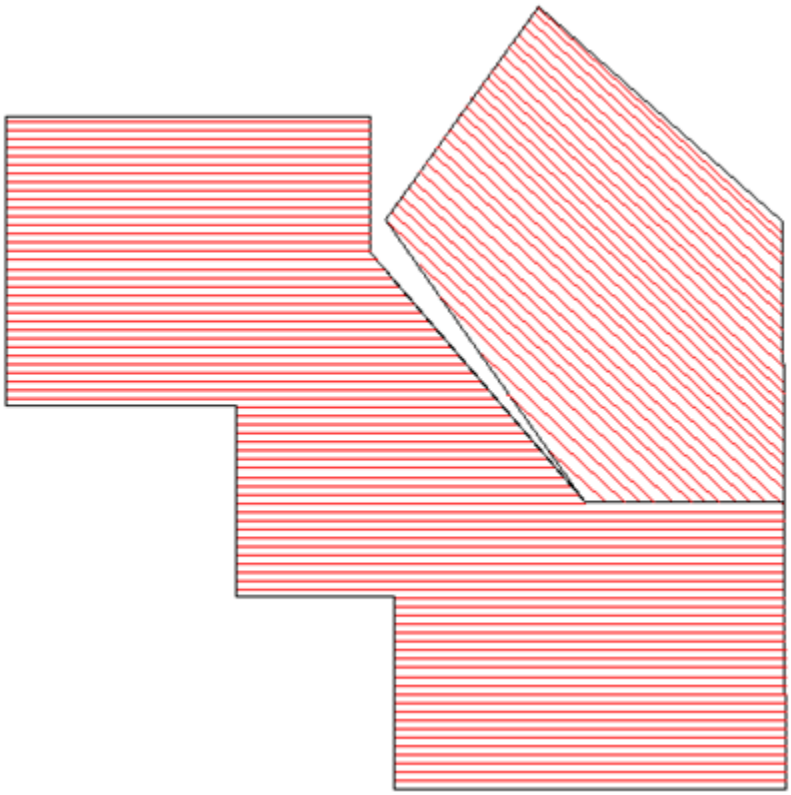


Figure 30. A more optimal solution to Fabre et al's field example.

Figure	Field	Area (acres)	Number of Points	Time (in milliseconds)	Savings
22	Iowa field 1	154.35	94	7807	Eliminates 5 turns
23	Iowa field 2	144.15	194	8651	0. Saves on headlands
24	Iowa field 3	54.45	87	1969	503.01 meters
25	North Dakota field 4	33.32	50	5016	0. Matches what a farmer would have done
26	Ohio field 1	Unknown	11	873	206.71 feet
27	Ohio field 2	Unknown	32	3234	0
29	Fabre field example	Unknown	14	834	-61 feet (pass width of 20') 0 feet (pass width of 30')

Table 3. Summary of results

CHAPTER 5. CONCLUSION

The algorithm presented in this thesis has been shown to be an efficient algorithm for finding the optimal solution for covering the field. QuickOPP yields results similar to those produced by Jin (2009), and on fields tested, produced no worse results than approaches taken by the farmers themselves. More importantly, the algorithm is significantly faster than previous algorithms proposed for solving the complete coverage path planning problem in an agricultural environment. Whereas prior algorithms took as long as $O(n^3 \log (n))$ time, the algorithm proposed here runs in $O(n^2)$ time. The approach of Jin (2009) suffered from significant performance problems once the boundary size exceeded 20 points and interiors were added. Since most field boundaries tend to have more than 20 points, the algorithm was not yet ready for use in the real world. The algorithm presented here has no such boundary size limitation, being capable of handling large fields with hundreds of points. All tests were run on a laptop with a 2.80 GHz Intel Core 2(™) Dual CPU, processor with 2.96 GB of RAM. A solution was found for all fields tested, including those with hundreds of points and several interior boundaries, in less than 60 seconds.

There are some limitations, however, with this algorithm and areas for further research. First, there are some scenarios where the first phase of the algorithm chooses a sweep direction that is not as good as it could be. This typically only happens, though in fields such as that depicted in the lower right corner of Figure 26 where there is a long narrow strip in one direction, and the rest of the field is fairly compact in a different direction. The long narrow strip may force the orientation of the sweep line to be parallel to the long side. However, this can be easily handled without compromising the running time by also considering sweeping in two directions: first as calculated for the field as a whole, and second in the normal direction.

A second limitation has to do with the fact the gain in performance was achieved at the risk of potentially missing the true optimal solution. This occurred because the search space for ways to decompose the field was restricted to subdividing the field parallel to the initial optimal direction. The algorithm assumes that dividing lines in other directions would most likely yield sub-optimal coverage directions. There may, however, be a scenario where that is not the case,

depending on the orientation and location of interior boundaries, and this algorithm would not find the more optimal solution.

Several variations can be made to this algorithm to improve the optimality. Given that each run of the algorithm only takes seconds, instead of confining the sweep to the overall optimal direction, the sweep and merge phases can be repeated once for each edge of the field, sweeping normal to the angle of the edge. This will increase the running time to O (n^3). Alternatively (or in addition), during the merging phase instead of confining the comparisons to four principle pass directions, the optimal direction for a merged region can be calculated using the same brute force method used to determine the initial optimal direction. Treating the number of discrete intervals as constant, this will increase the running time by a constant. Finally, during the sweeping phase, the tolerance for determining whether a trapezoid should be ended and a new one started can be increased from 170° to the desired tolerance so as to reduce further the number of trapezoids generated. The tolerance can even be modified so as to avoid ending a region so long as the region remains convex.

Regarding areas for further research, this algorithm only operates in two dimensions. It does not take into account the effect of hills in finding the optimal direction. If the hills are particularly steep, then it would not be practical to travel in certain directions due to the slope. In addition, hills introduce additional costs such as soil erosion that is not accounted for in an efficiency/distance based cost formula. Finally, this algorithm does not solve the problem of coordinating the work of several machines in the field, nor does it account for determining when it would be most optimal to stop and refuel. Such an algorithm would be useful, for example, while harvesting fields. In that scenario, there is typically a harvester moving through the field harvesting, with a tractor alongside pulling a wagon into which the harvest crop is loaded. Eventually the wagon gets full and either needs to unload or be replaced by another wagon. Finding the most optimal path that allows for efficient loading and unloading of the following vehicle would also be useful, but is not accounted for here.

Bibliography

Atkar, P. N., H. Choset, A. A. Rizzi and E. U. Acar. Exact Cellular Decomposition of Closed Orientable Surfaces Embedded in R^3. In *Proceedings of International Conference on Robotics and Automation*, pp. 699-704, Seoul, Korea, 2001.

Barraquand, J., B. Langlois, and J-C Latombe. Numerical Potential Field Techniques for Robot Path Planning. IEEE Transactions on Systems, Man and Cybernetics, 22(2): 224-241 (1992)

Berg, M. D., M. V. Kreveld, M. Overmars and O. Schwarzkopf. *Computational Geometry*. 2nd ed. Springer – Verlag Berlin Heidelberg 1997, 2000.

Chibin, Z., W. Xingsong, and D. Yong. Complete Coverage Path Planning Based on Ant Colony Algorithm. In *15th International Conference on Mechatronics and Machine Vision in Practice*, pp. 357 – 361, Auckland, New Zealand. December 2008.

Choset, H. Coverage of Known Spaces: The Boustrophedon Cellular Decomposition. *Autonomous Robots*, 9: 249-253, 2000.

Choset, H. and P. Pignon. Coverage Path Planning: The Boustrophedon Cellular Decomposition. In *Proceedings of International Conference on Field and Service Robotics*, Canberra, Australia, 1997.

Dorigo, M and L.M. Gambardella. Ant colonies for the traveling salesman problem. *Biosystems* (1997).

Du, X, H. Chen and W Gu. Neural Network and Genetic Algorithm Based Global Path Planning in a Static Environment. *Journal of Zhejian Univeristy SCIENCE* 64(6): 549-554, 2005.

Fabre, S., P. Soures, M. Taix and L. Cordesses. Farmwork path planning for field coverage with minimum overlapping. In *Proceedings of the 2001 IEEE International Conference*, pp. 691-694, 2001.

Gamma, E., R. Helm, R. Johnson, and J. Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wessley (1994).

González, E., O. Álvarez, Y. Díaz, C. Parra, and C. Bustacarra. BSA: A Complete Coverage Algorithm. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pp. 2040-2044, Barcelona, Spain, April 2005.

Guo, Y and M. Balakrishnan. Complete Coverage Control for Nonholonomic Mobile Robots in Dynamic Environments. In *Proceedings of the 2006 IEEE International Conference on Robotics and Automation*, pp. 1704-1709, Orlando, Florida, May 2006.

Jin, J. and L. Tang. Optimal Path Planning for Arable Farming. *2006 ASABE Annual International Meeting*, Paper No. 0611581-12, Portland, Oregon, 2006.

Jin, J. Optimal Field Coverage Path Planning on 2D and 3D Surfaces. *A dissertation submitted to the graduate faculty in partial fulfillment of the requirements for the degree of Doctor of Philosophy, Agricultural Engineering*. Iowa State University, Ames, Iowa, 2009.

Kang, J. W., S. J. Kim, M. J. Chung, H. Myung, J. H. Park and S. W. Bang. Path Planning for Complete and Efficient Coverage Operation of Mobile Robots. In *Proceedings of the International Conference on Mechatronics and Automation*, pp. 2126-2131, Harbin, China, 2007.

Martin, P., and A.P. del Pobil. Application of artificial neural networks to the robot path planning problem. *Transactions on Information and Communications Technologies*. Vol 6, 1994. WIT Press.

Meuth, R. J. and D. C. Wunsch II. Divide and Conquer Evolutionary TSP Solution for Vehicle Path Planning. 2001.

Oksanen, T. and A. Visala. Path Planning Algorithms for Agricultural Machines. *Agricultural Engineering International: the CIGR Ejournal.* Manuscript ATOE 07 009. Vol. IX. July 2007.

Noguchi, N. and H. Terao. Path Planning of an Agricultural Mobile Robot by Neural Network and Genetic Algorithm. *Computers and Electronics in Agriculture* 18: 187-204, 1997.

Rimon, E. Exact Robot Navigation Using Artificial Potential Fields. *IEEE Transactions on Robotics and Automation*, 8(5): 501-518 (1992)

Ryerson, A. E. F. and Q. Zhang. Vehicle Path Planning for Complete Field Coverage using Genetic Algorithms. *Agricultural Engineering International: the CIGR Ejournal.* 9: Manuscript ATOE 07 014, July 2007.

Sörensen, M. J. Artificial Potential Field Approach to Path Tracking for a Non-Holonomic Mobile Robot. In *11th Mediterranean Conference on Control and Automation*, June 2003.

Shah-Hosseini, H. The intelligent water drops algorithm: a nature-inspired swarm-based optimization algorithm. In *International Journal of Bio-Inspired Computation*, 1(1/2): 71-79, 2009.

Sorensen, C. G., T. Bak, and R. N. Jorgensen. Mission Planner for Agricultural Robotics.

Tse, P.W., S. Lang, K.C. Leung and H.C. Sze. Design of a navigation system for a household mobile robot using neural networks. In *Proceedings of the International Conference on Neural Networks*, Anchorage, AK, 1998, pp 2151-2156.

Yang, S. X. and C. Luo. A neural network approach to complete coverage path planning. IEEE Transactions on Systems, Man, and Cybernetics – Part B: Cybernetics. 34(1): 718-725, 2004.

Zelinsky, A., R.A. Jarvis, J.C. Byrne and S. Yuta. Planning Paths of Complete Coverage of an Unstructured Environment by a Mobile Robot. *In Proceedings of International Conference on Advanced Robotics.* 1993.

Zhang, C, X. Wang and Y. Du. Complete Coverage Path Planning Based on Ant Colony Algorithm. In *Proceedings of the 15th International Conference on Mechatronics and Machine Vision in Practice,* pp. 357-361, Auckland, New Zealand, December 2008.