STACKER CRANE PROBLEM STATE SPACE REDUCTION

BY

Frank Kreimendahl

BS in Mathematics, University of Rochester, 2007

THESIS

Submitted to the University of New Hampshire
in Partial Fulfillment of
the Requirements for the Degree of

Master of Science

in

Computer Science

April, 2013

This thesis has been examined and approved.

Thesis director, Wheeler Ruml,
Associate Professor of Computer Science

Michel Charpentier,
Associate Professor of Computer Science

Roger Grinde,
Chair, Decision Sciences Department and Associate Professor of Management Science

Date

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

STACKER CRANE PROBLEM STATE SPACE REDUCTION

by

Frank Kreimendahl

University of New Hampshire, April, 2013

Many resource delivery problems, from delivery vehicle routing to circuit board assembly, can be expressed as stacker-crane problems (SCPs). In SCPs, a single agent must transfer resources from their starting locations to goal locations by traversing a graph. These problems have been split into many categories based on their graph structure and other problem properties. This thesis examines preemptive stacker-crane problems on four-connected grids, which are common graphs in motion planning domains. A wealth of research has already been done on motion planning on graphs, both for optimal and suboptimal solutions. This thesis focuses on reducing the number of vertices in the graph and the number of actions available at each vertex rather than presenting a specific solution, which allows for any solver to be used on the reduced graph.

Two specific reductions are proposed in this thesis. Both are proven to preserve an optimal solution so that a solution on the reduced graph can be trivially converted to a solution on the original graph. The first reduction limits movement to a Hanan grid and resource drops to Hanan points. The second reduction recursively removes redundant corners, which is shown to preserve optimal paths also. This technique recognizes path symmetries and seeks to break them without compromising optimal solutions.

CHAPTER 1

Introduction

Planning in AI is difficult. In many planning domains, there are too many possible states to explicitly instantiate them all. As a result, planning algorithms build graphs of states, where vertices represent individual states in the domain and edges represent possible state transitions. Rather than building a full graph before looking for solutions, algorithms grow the graph by applying actions to a vertex, resulting in a new state in the graph or a new edge to an existing state. This state space exploration process continues until a goal state is reached. Many actions may be applicable in each state, there are many states, and many paths through the graph (which correspond to sequences of actions) may reach each state. It is difficult to find reasonable action sequences in such large graphs without being clever about which actions to explore to achieve a goal. Reducing the number of states in the state space can speed up a solver drastically.

1.1 The Problem

The stacker-crane problem (SCP): Given a graph $G = (V, E, A, C_i)$, V is a set of vertices, E is a set of edges with specific traversal costs, A is a set of $(start, goal)$ vertex pairs that must be visited in that order, and C_i is an agent's starting vertex. The goal is to find a minimal-cost sequence of edges that start at a specific start vertex, C_i and traverse all of the arcs in A. This corresponds to a single agent, in this case a crane, moving resources from their starting locations to specified goal locations while minimizing the total distance that the crane has traveled. The $(start, goal)$ vertex pair for a resource R will be denoted as (R_s, R_g).

In the version of the problem that we consider here, there are several further specifications. First, the crane can only transport one object at a time. A vertex may hold any number of resources and those resources are not ordered. Therefore, the crane can pick them up from the vertex in any order. The crane is confined to a rectangular four-connected grid, and its actions at any vertex are a subset of: up, down, left, right, pickup, and drop. Because of the nature of the graph, horizontal and vertical movements, and therefore distance, are independent. Measuring distance between two vertices on this graph is called the Manhattan distance, and the distance between vertices v_1 and v_2 is denoted as $\|v_1 - v_2\|_1$. This is equal to $|v_1.x - v_2.x| + |v_1.y - v_2.y|$. The pickup action is only valid if the crane is on the same vertex as a resource and the crane is not already carrying a resource. The drop action is only valid if the crane is carrying a resource. Some move actions may be invalid if the crane is at the edge of the graph (if we assume that the graph does not extend infinitely). The crane may temporarily drop resources at intermediate locations if that action decreases the overall distance that the crane has to travel compared to carrying the resource all the way to its goal. These variations on the original problem require a slightly different problem formulation. Since the graph is four-connected, E can be generated from V. Allowing intermediate drop locations for resources, called preemption, changes the requirements for A. Resources must be picked up from their start and dropped at their goal, but they may be dropped and picked up again at other vertices before they reach their goal. Therefore, the problem can be specified on $G = (V, A, C_i)$.

An example stacker-crane problem with two resources, R and T, is shown in figure 1-1. The solution to the problem is a sequence of actions that the crane will execute to move R to R_g and T to T_g. To get a better understanding of the size of the state space and why we want to lazily generate its graph, we can examine this example. Each state consists of information about all objects (location, crane state, etc.) that an action may change. In this problem, actions can change the location of the crane, what the crane is carrying, and the location of a resource. If we consider possible configurations for crane and resource

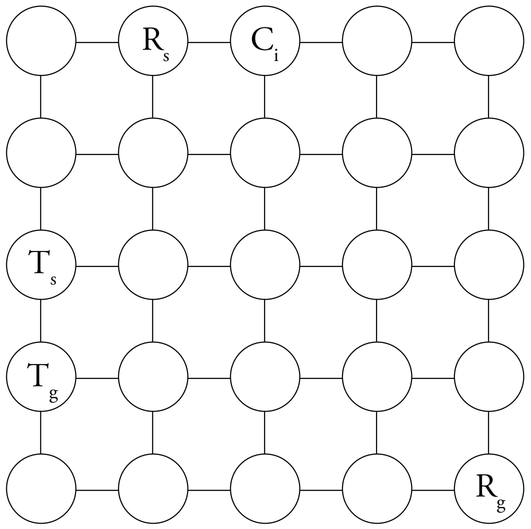


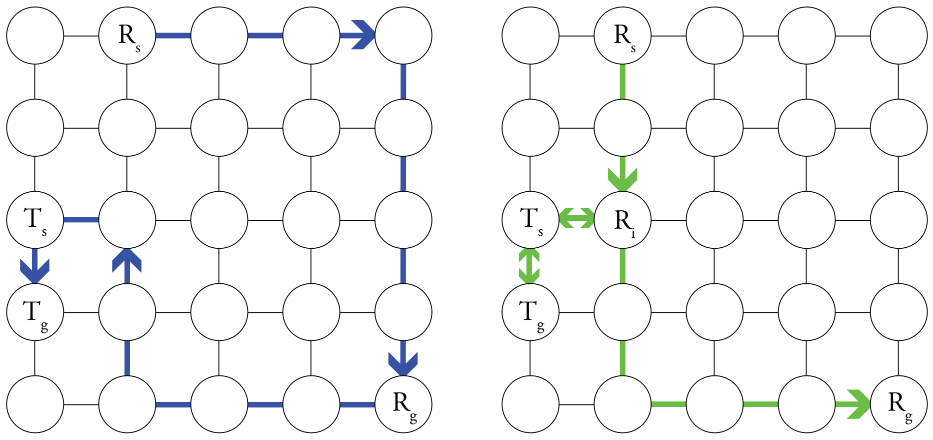
Figure 1-1: Example stacker-crane problem on a four-connected graph.

location, the problem in figure 1-1 has 25 vertices, and the locations of R, T, and C are all independent. This means there are $25 \times 25 \times 25 = 25^3 = 15,625$ states. This does not account for the crane carrying a resource. The crane can hold a resource at any vertex, and other resources may be anywhere on the graph for each crane position. This adds an additional $25 \times 25 = 625$ possible states, for a total of $16,250$ states. We can find the general equation by counting the number of states in which the crane is empty and the number of states in which the crane is carrying a resource separately. For a problem with k resources on a $m \times n$ grid, the number of possible states is $(m \times n)^{k+1} + (m \times n)^k$. The first term results from $k+1$ independent objects (k resources and the crane) that can lie anywhere on the grid. The second term assumes that crane is carrying one resource, so there are k independent objects on the grid. If we double the grid's dimensions to be a 10×10 grid, the new problem has $100^3 + 100^2 = 1,010,000$ states. If, instead of changing the grid size, we add one more resource, that would result in $25^4 + 25^3 = 406,250$ states. 8 resources on a 100×100 grid yields over 10^{32} states! This is not something that currently fits in the memory of any computer. Many of the states are irrelevant for finding a solution, so we wouldn't want to generate those states anyway. The solution will ultimately be an action sequence that transitions through some of these states. This is a tremendous combinatorics problem if we approach it in a naïve manner.

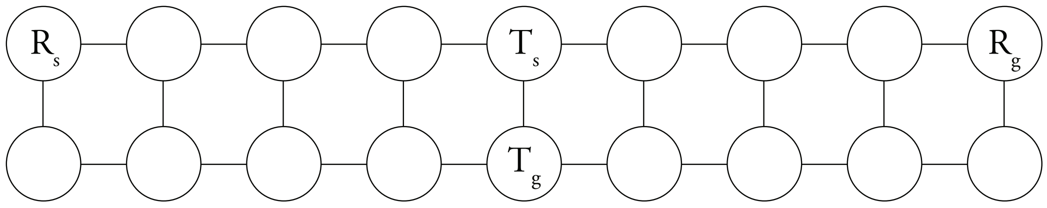
In this domain, we are trying to minimize the total cost of the crane's actions to achieve all goals. The crane may drop off a resource at an intermediate location in order to make the entire solution cost smaller, even though it makes the cost for transporting that single resource larger. With each intermediate vertex, the number of possible sequences to achieve all goals increases drastically. In a four-connected graph, there are so many intermediate points that it is difficult to find optimal solutions, even for small problems (3 resources on a 100x100 graph) using standard heuristic-guided search techniques. This result comes from the difficulty in determining whether an intermediate drop point is along an optimal sequence or not until much later in the sequence. This means that many actions after non-optimal drop actions are considered, and many optimal partial plans are considered when only a single plan is needed. This computation wastes both memory and CPU time, and may result in memory limits being reached before a solution is found. This thesis shows how to reduce the number of states and actions considered to find an optimal solution. **My thesis proves that we only need to consider a small subset of all vertices on the graph to find an optimal solution.**

1.2 Importance

The SCP captures the essence of an important issue in pick-and-place domains – domains that involve a single agent moving objects around. Making the simplification that an object can only be dropped in its goal location may lead to longer (sub-optimal) motion plans, or may lead to no solution. Therefore, we want to consider dropping an object at intermediate locations and returning to it later. Consider a domain where a robot is asked to move a box through a door and the door is currently closed and blocked by the box. To solve the problem, the robot must move the box out of the way, open the door, and then pick up the box and move through the doorway. The robot cannot solve the problem without dropping the box in an intermediate location. Dropping objects at an intermediate location allows for solutions to more problems than only carrying objects straight to their goal. A task as



(a)



(b)

Figure 1-2: Optimal paths for non-preemptive SCP (top left) and preemptive SCP (top right). Example SCP where preemption is very advantageous (bottom).

simple as swapping two objects requires dropping the same object twice.

Drawing on ideas from the path-planning literature, A* [Hart et al. (1968)] and its many extensions can be applied to SCP. Considering a drop action at every possible intermediate location can require prohibitively large memory resources, however. Even with strong heuristic guidance, many unnecessary drop locations will be considered for each task ordering. Though many of these drop points lie along an optimal action sequence, we will see that a lot of the drop locations are redundant. Reducing the graph will remove many potential drop locations, leaving only a few that can lead to an optimal solution. This, in turn, leads to a smaller state space and faster solution times.

1.3 Choosing the Domain

How important is preemption? Without preemption, an SCP on a four-connected graph can be converted quickly to a complete graph of important vertices with edge costs based on their rectilinear distances. None of the intermediate vertices serve any purpose because dropping a resource on them is not allowed. Thus, a grid is reduced to a complete undirected graph with no more than $2n$ vertices for n resources. Using the reductions presented in this thesis, a preemptive graph has a maximum of $(2n)^2$ vertices for n resources (based on the number of vertices in the corresponding Hanan grid). There is already a wealth of research available to solve the non-preemptive SCP problem on a graph.

Although adding preemption makes finding solutions on a four-connected graph harder as many more vertices must be considered, solutions costs to the preemptive SCP can be considerably smaller than their non-preemptive counterparts. Figure 1-2a shows optimal solutions on the example that we have used throughout the thesis. On the left is an optimal solution for a non-preemtive SCP. The crane's total pathlength is 14 units. On the right is one of the optimal paths for a preemptive SCP given the same resource starts and goals. Its pathlength is 11 units, 20% shorter than the non-preemptive case. Figure 1-2b shows a simple case where preemption can decrease the crane's total pathlength by nearly 50%.

1.4 Approach

This thesis starts, as is already evident, with a chapter that briefly introduces the stacker-crane problem and hints at an approach to reduce its state space. The next chapter discusses the results of related works, both presenting their conclusions and highlighting differences. Chapter 3 is the pith of the thesis. It presents state space reductions on an SCP's initial four-connected graph that maintain at least one optimal solution. Each step is rigorously proven to guarantee that optimality is preserved. In chapter 4, experimental results, along with discussions about the reductions, limitations to this model, and possible extensions appear. Chapter 5 concludes the thesis.

CHAPTER 2

Previous Work

Many variations on the Stacker-Crane Problem (SCP) have been formulated and analyzed since it was first introduced by Frederickson et al. (1976). An overview of several SCPs will be presented below. Several variations in graph structure and preemption lead to interesting differences in properties between different types of SCPs. Berbeglia et al. (2007) provides a classification scheme of delivery problems based on problem statements. In that scheme, the SCP is one of the problems with the most rigid specifications in the problem hierarchy. Changing constraints such as the number of cranes, capacity of cranes, and allowing variable capacity requests all generalize the SCP. The first section of this chapter will focus specifically on solving the SCP on various graph topologies. All of these variations assume a single crane that can carry a single resource at a time, and that there are constant-size resource requests. The SCPs discussed will vary in graph properties and are discussed as both preemptive and non-preemptive problems. The second section will present other work on grid pathfinding.

2.1 Stacker-Crane Problem on a graph

The most general SCP is presented on a graph. This section presents previous results of both non-preemptive and preemptive SCPs on graphs.

2.1.1 Non-preemptive SCP on a graph

The Stacker-Crane Problem was named and first presented and analyzed by Frederickson in 1976. The initial formulation required a set of arcs to be traversed in any order over a complete graph. Each arc is a single directed edge in the graph. Three variations on this problem are presented: V_s (the starting vertex of the crane) and V_g (the goal vertex of the crane) are the same, V_s and V_g are different, and V_g is unspecified. The first is the most similar to a circuit in the traveling salesman problem. To prove its computational complexity, SCP is reformulated as a recognition problem: given $C \in \mathbb{N}$, decide whether a solution to the SCP exists with $cost(solution) \leq C$. It is shown that the traveling salesman problem can be reduced to the SCP in polynomial time and a solution to the SCP can be verified in polynomial time. Therefore, the circuit version of SCP is NP-complete. Frederickson further shows that the other two variations are NP-complete by simple reductions. The third version, in which the crane can stop at any vertex, is the version that we are most interested in. The paper also presents an approximation algorithm for the general SCP that requires a fully connected graph. The graph is processed so that its edge costs follow the triangle inequality and so that each vertex is either a start or goal vertex for a single arc. This is possible by splitting vertices that are serving as more than one arc endpoint and excising vertices that are not an endpoint of any arc. The approximate solution is at most $\frac{9}{5}$ larger than the optimal solution and the algorithm runs in polynomial time.

Since this formulation is fully connected and follows the triangle inequality, the edge between each start and goal vertex has the lowest cost and is the only edge considered in delivery. Without considering arcs that span more than one edge, every resource is moved directly from its start to goal vertex. If no resource moves along more than one edge, preemption is not possible.

The motivation for the solution to this problem is given briefly as "practical applications such as operating a crane or forklift, or driving a pick-up and delivery truck." The paper makes no more mention of any physical problems that its proposed solutions may be applied to. Frederickson focuses more on the theoretical properties of the SCP than on what problems it might help solve.

2.1.2 Preemptive SCP on a graph

A rigorous analysis of preemptive SCPs on a graph did not appear until recently [Kerivin et al. (2012)]. Kerivin et al. focus on a problem related to SCPs: the Single-Vehicle Preemptive Pickup and Delivery Problem (SPPDP). In the general SPPDP, there is still a single delivery agent, but it is not limited to carrying one resource at a time. In addition, different resources may be requested in different amounts. It is possible for resource R to take up twice as much space as resource T in the vehicle, rather than in SCPs, where all resources are the same size. The SPPDP is specified on a directed graph which may be incomplete, and each vertex is at the end of at most one resource arc. The paper considers the unitary SPPDP, in which the delivery agent only has space for a single resource. The only difference between the unitary SPPDP and the preemptive SCP is that the agent in the unitary SPPDP is only allowed to traverse each directed edge at most once. The paper proves that, for graphs with vertices that follow the SPPDP restrictions of no more than one arc endpoint per vertex, an optimal solution exists when traversing each directed edge at most once. An integer linear program solution is presented for the SPPDP that can easily be adapted to the preemptive SCP. Like the case of non-preemptive SCPs on a graph, the traveling salesman problem can be reduced to a preemptive SCP, suggesting that it is also NP-complete [Anily et al. (2006)].

2.2 Stacker-Crane Problem on a line

Rather than Frederickson's introduction of the SCP on a graph, the presentation of the SCP along a single line [Atallah and Kosaraju (1988)] has a very specific motivating application that spurred analysis of the problem. Atallah et. al. consider a robot with a fixed rotating base and a telescoping arm with a gripper at the end. The problem is presented as a set of objects that the robot must pick up and move to specific locations. The two movement actions are rotating the arm and telescoping the arm. These two actions are split into different subproblems and considered independently. Considering an SCP with only a

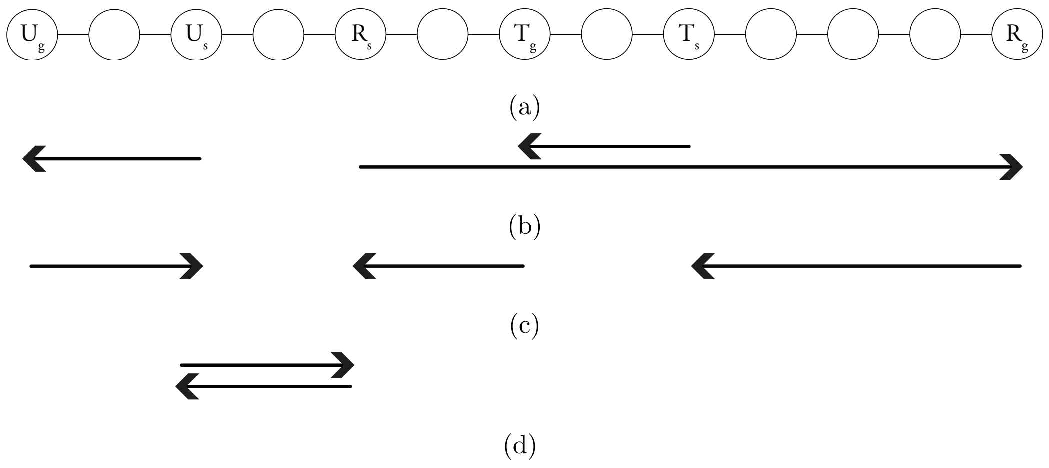


Figure 2-1: Example of arc augmentation on non-preemptive SCP on a line.

telescoping arm allows for movement only along a single line.

2.2.1 Non-preemptive SCP on a line

The non-preemptive stacker-crane problem was introduced independently in two papers [Atallah and Kosaraju (1988), Ball and Magazine (1988)]. Atallah's paper considered a telescoping robot arm moving objects back and forth on a table. Ball's paper is motivated by automated chip placement on circuit boards. In this second paper, preemption is ignored because the chips are being moved from a line of feeder trays to their sockets and cannot be stored at intermediate locations on the circuit board. Even though the problem is over a two dimensional space, it is assumed that the arm will be moving sequentially to each feeder tray without returning to a previous one. When this is the case, any loop in the arm's path is a fixed distance (from a feeder tray to a goal and back to the feeder tray) and can be ignored. Ball presents a polynomial-time algorithm that provides bounded suboptimal solutions. On a Manhattan metric, the solution is optimal because of the independence of vertical and horizontal movement.

The Atallah paper considers a properly one-dimensional telescoping arm moving objects along a line. An example SCP on a line is shown in figure 2-1, with arcs added for two arc augmentation steps. Since preemption is not considered, there is no advantage for any detours and any solution will contain all arcs in the problem. These arcs are laid out over the line, as shown in figure 2-1b. It is not guaranteed that each goal of an arc is the start of the next, so the required arcs must be augmented with additional edges in order to have a fully connected tour. This is split into two steps. In the first step, all edges that are traversed a different number of times in the two directions have antiparallel edges added to balance out as every vertex must have the same number of edges leaving as it has coming in (figure 2-1c). An unbalanced number of edge traversals implies that the arm stopped somewhere along its route. This gives valid minimal paths for any subset of the line that is covered by an arc, but there may be disjoint sections in between covered segments. To make a feasible motion plan that reaches all of the section, minimal edges must be added to connect all of the sections, as seen in figure 2-1d. The resulting tour is optimal, since the graph is linear and each subsection is optimal. These augmentations are both fast, and the running time of Atallah's algorithm is $O(m + n\alpha(n))$, where m is the number of objects, $\alpha()$ is the inverse Ackermann function, and n is the number of vertices that may be start or goal locations for the objects.

2.2.2 Preemptive SCP on a line

The robotic arm in Atallah's paper can both rotate and telescope. He proves directly that a preemptive SCP on a circle can be solved in polynomial time. The solution is easily applicable to a preemptive SCP on a line, so preemptive SCPs on lines can be optimally solved in polynomial time. This can be seen by embedding the line on a small arc of a circle, in which a full rotation is possible but the cost is too high to be feasible. The solution for both preemptive and non-preemptive circular SCPs will be discussed in the next section.

2.3 Stacker-Crane Problem on a circle

As mentioned in the previous section, Atallah et. al. analyzed SCPs on a circle with the intention of planning a rotating robot's movements. The paper finds that both preemptive and non-preemptive SCPs can be solved in polynomial time. The proof that a preemptive SCP on a circle can be solved in polynomial time can be easily extended to a preemptive

SCP on a line.

2.3.1 Non-preemptive SCP on a circle

The solution for a SCP on a circle takes a similar approach to a SCP on a line. A large difference, however, is that it is no longer true that the number of traversals in both directions along each edge must be balanced because it is possible to fully rotate around the circle. Instead of having a balanced number of traversals along each edge, there must be the same number of unbalanced traversals along each edge. For example, if edge e_1 has 2 clockwise traversals and 4 counterclockwise traversals, a different edge e_2 must have 2 more counterclockwise traversals than clockwise. With this rule, the solution uses the same two-step augmentation process as the solution on a line. First edges are added to the required arcs so that all of the edges have the equal differences in traversals. Second, additional augmenting edges are added with the observation that there is always an edge that has at most one augmenting edge. The computation for an optimal solution runs in $O(m+n\log n)$ time.

2.3.2 Preemptive SCP on a circle

In the case of a preemptive SCP on a circle, it shown that each object needs to be moved in only one direction to reach its destination. Furthermore, at most one object must be transported the long way around the circle to its destination. With proofs of these two theorems, the calculation time for a solution is shown to be $O(m+n)$

2.4 Four-connected Grids

The goal of this thesis is to present a state space reduction of a four-connected grid. Although the stacker-crane problem has been studied on many topologies, it has not been considered on four-connected grids. Four-connected grids are popular in video games, as well as being used as discretizations of rectangular rooms in robotic motion planning. Recently, symmetry-reducing methods have been considered for path finding on video game-style grid

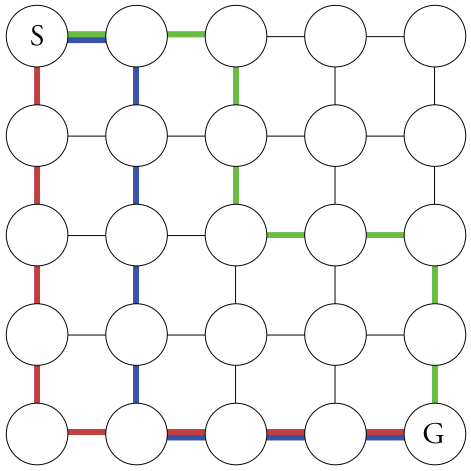


Figure 2-2: Three of the many minimal paths from S to G.

maps.

Harabor and Botea (2010) use the intuition that there are many equivalent paths on a four-connected grid and that parts of the grid can be pruned. Choosing the prunings wisely still allows for optimal solutions, and the pruning process can also be completed quickly. In the maps that Harabor analyzes, there are multiple rooms connected by thin corridors. Interior vertices in areas with no start or goal vertices are pruned, and the gaps in between are converted to 'macros' - single movements that cover multiple horizontal or vertical edges. This results in paths that follow close to obstructions whenever possible, and only make a few jumps across open spaces. The pruning algorithm is tailored to speed up finding a single path using an A* search. The paper finds that this pruning works best for grid maps with large open areas, as only a small fraction of vertices need to be considered. It provides a speedup factor between 2 and 3 for finding single paths through the reduced search space, including the reduction time. We will see a similar intuition for pruning nodes to reduce the SCP state space. Figure 2-2 shows three possible paths through a graph from a start vertex to goal vertex. While it is an interesting combinatorics problem to determine how many optimal paths there are, we are satisfied with finding a single one. We are looking for any simplifications that we can find to faster arrive at an optimal answer.

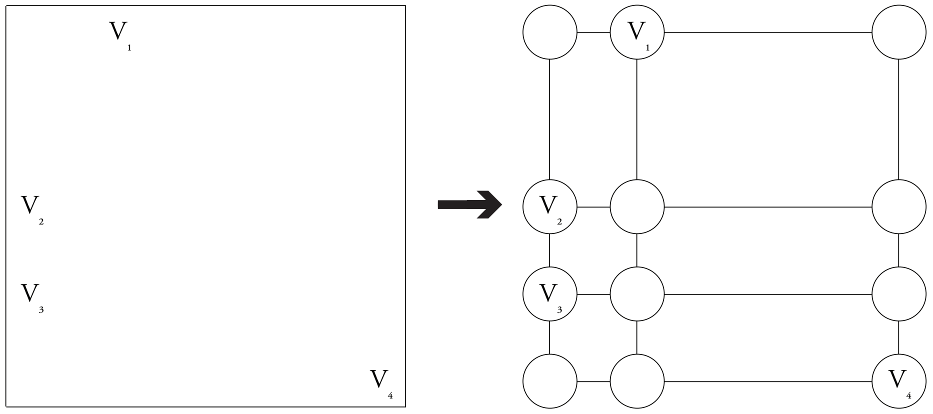


Figure 3-1: Example of a set of vertices (left) and the resulting Hanan grid (right).

CHAPTER 3

Reductions

How can we pick a subset of intermediate points that is guaranteed to contain an optimal solution, but as small as possible in order to shrink the state space? This chapter presents incremental steps to decrease the state space of a SCP. Each step is proven to maintain at least one optimal solution to the original problem. First, a quick summary of the steps is given. Next, these reduction steps are presented in detail with supporting proofs. These reductions are based on a specific property of the SCPs that we are considering: the graph that the crane can move over is a four-connected graph.

3.1 Overview

In lemma 1, we will show that we can ignore any points outside the bounding box of the entire set of start/goal positions because it will never be advantageous to leave that boundary, let alone drop a resource there. Additionally, we show in lemma 4 that we can restrict movement to a Hanan grid. A Hanan grid is a set of points with horizontal and vertical lines through each point. An example Hanan grid is shown in figure 3-1. Using start and goal locations as the set of points to construct the grid, the lines form edges that are possible crane paths. The grid is contained within the bounding box, so every crane position on it is acceptable. We can, in fact, limit our search to only crane positions on the grid. We know that our starts and goals all lie at cross points of the grid because we constructed the grid that way. With all start and goal points lying only at the corners of grid cells, there is no advantage to cutting diagonally across a grid cell. Because the crane moves over a four-connected graph, distance between two vertices is measured with Manhattan distance. Therefore, moving along the cell's edges is the same distance as any diagonal path. We do not have a risk of discounting all optimal solutions with these restrictions even though we have reduced the search space. We can further look at where we will consider drop points within this grid. Lemma 6 proves that if the crane is carrying a resource along an edge in the Hanan grid, its path will contain both of the endpoints of that edge. If the crane drops off a resource along this edge, it could also drop the resource off at one of the endpoints without increasing the total cost of the path.

These reductions leave us with a Hanan grid, which is a subset of the original graph, as shown in theorem 7 and lemma 8. A further reduction, theorem 9, allows us to remove vertices that are provably redundant. Any optimal solution that passes through these removed vertices can be modified to pass through other vertices while maintaining optimality.

3.2 Proofs

The following lemmas and theorems reduce the search space incrementally. They assume the search space starts with a four-connected undirected graph with unit-cost edges and allow for the drop action at any vertex. Resources don't interfere with each other, so multiple resources may occupy the same vertex and the crane can pick up any resource at its vertex (assuming that the crane is not already carrying a resource). By proving that an optimal solution exists with the restrictions in the following theorems, we can effectively reduce the size of the graph we need to search over for an optimal solution. Since we are reducing the

```
pathThroughVertex($V, R_s, R_g, v$)
 1. Move horizontally from $R_s$ to closest vertex to $v$
 2. Move vertically to other side of bounding box
 3. Move horizontally to $R_g$
```

Figure 3-2: An algorithm that finds an optimal path from R_s to R_g that passes through v.

graph size rather than providing a specific solution to the SCP, many algorithms from the existing literature can be applied to the updated graph.

3.2.1 Hanan Grid Reduction

Definition 1. Bounding box

A bounding box for two vertices, v_1 and v_2, is the set of all vertices contained in the rectangle formed by horizontal and vertical lines passing through v_1 and v_2. The bounding box contains boundary vertices, so the bounding box is never empty. For a given set of points, the bounding box is unique. The bounding box for vertices v_1 and v_2 are referred to as the "bounding box of (v_1, v_2)".

Lemma 1. *There exists an optimal solution within the smallest bounding box that contains the start and goal locations of all resources.*

Proof. Consider the smallest bounding box, B, that contains all start and goal locations of all resources. If the crane moves outside of B, each movement away from B must be paired with a movement toward B in order to reach any resource's start or goal location. This movement costs strictly more than an equivalent path that does not consider moving outside of B. Therefore, the crane never considers moving outside of B. □

Lemma 2. *For all vertices in the bounding box of resource R, there is an optimal path from R_s to R_g that passes through that vertex.*

Proof. This lemma can be proven by creating an algorithm that constructs a path from R_s to R_g and proving that the algorithm always passes through v. Because horizontal and vertical distance measurements are independent, the crane can interleave horizontal and vertical steps toward R_g in any order. The simple algorithm in figure 3-2 constructs a path that will always be optimal and will always pass through the desired vertex v, as long as v falls within the bounding box. In line 1, the crane moves to the closest vertex to v that it can through only horizontal movement. At this point, the crane is not displaced horizontally from v. The crane moves vertically to the opposite side of the bounding box in line 2. The crane passes through vertex v in this step because the only possible displacement between the crane and v after line 1 is vertical, and the crane's movement passes through every vertex in the bounding box on this vertical line. The crane starts the last step on the same horizontal line as R_g because it crossed the entire bounding box from the horizontal line that passes through R_s. It moves the rest of the horizontal displacement to the goal in line 3. In the edge cases in which v is on the same horizontal or vertical line as R_s or R_g, one of the steps will have an actual movement of 0. The algorithm still works in these cases. If R_s and R_g are on the same line, the crane must necessarily pass through every vertex in the bounding box to reach R_g. The algorithm in figure 3-2 finds a path from R_s to R_g that passes through v without specifying where v falls in the bounding box.

We must still prove that the path determined by the algorithm is optimal. This is easy to show by examining the horizontal and vertical distances that the crane covers. The only vertical movement is in line 2, which means that the crane has travels the least distance possible, vertically. In line 1, the crane does not move away from R_g, because that would cause it to leave the bounding box and v is inside the bounding box. The crane reaches R_g in line 3, so it cannot possibly be moving away from R_g. We haven proven that the crane moves from R_s to R_g in the three lines of the algorithm. Therefore, the total horizontal distance is covered in lines 1 and 3. The crane does not move away from R_g in either of those steps, so the crane must also be covering the smallest horizontal distance. If the crane covers both the smallest horizontal and vertical distance possible using a Manhattan metric,

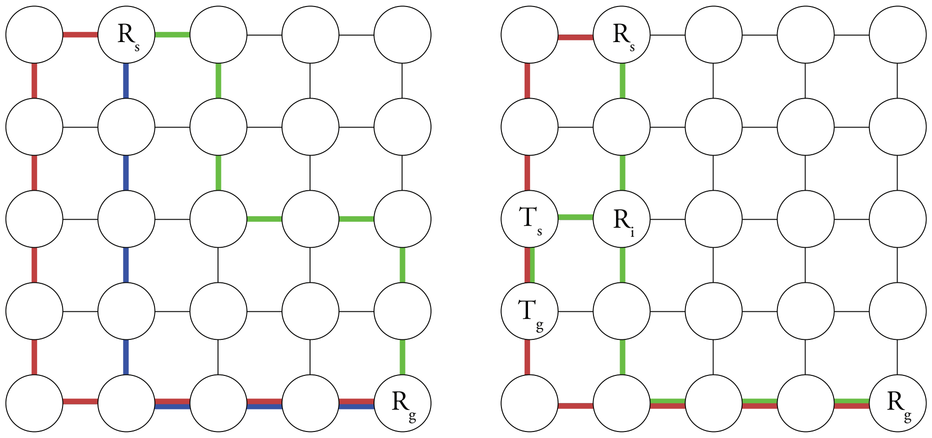


Figure 3-3: Example of lemma 3: Case 1 (left) and case 2 (right).

we know that the path is optimal. This satisfies the assertion of the lemma. □

Lemma 3. *For all resources R, with start locations R_s and goal locations R_g, do not consider moving R outside of the bounding box of (R_s, R_g).*

Proof. Figure 3-3 shows specific examples of the two general choices: R will be dropped off at R_g (case 1), and R will be dropped off somewhere else (case 2). In case 1, any movement outside of the bounding box requires the opposite movement back toward the bounding box before R_g is reached. The minimal distance is simply $\|R_s - R_g\|_1$ and a path with that distance can be easily found within the bounding box as shown by lemma 3-2. The blue and green paths in case 1 of figure 3-3 show two such paths. Also illustrated is a red path that leaves the bounding box. On the red path, the crane's initial movement brings it outside the bounding box. This movement is matched with a later movement that returns the crane to the bounding box. Each time there is a matched movement like this, the path length is unnecessarily increased. Therefore, a strictly shorter path can be found without leaving the bounding box.

In case 2, R is dropped off at an intermediate location, R_i. This means that the crane is going to move at least one other resource and must later return to R_i to continue moving R to R_g. We will call the initial position of the first other resource O_s, and O_g is the last position of the resource the crane drops before returning to R_i. If the crane moves R

outside its bounding box, the crane must later move R back to the bounding box (since R_g is contained in the bounding box). The crane can instead drop R off on the boundary of the bounding box and return to it without increasing its total path length. The crane's full path must go from R_s to R_i, R_i to O_s, O_s to O_g, O_g back to R_i, and R_i to R_g. Note that some of these distances may be 0. We can show that picking the closest vertex to O_s inside R's bounding box is minimal by breaking each of these paths into their component parts and examining them. No matter where R_i is, we must move from R_s to R_i and R_i to R_g. Since R_i is inside the bounding box, we know we can find an optimal path through it, using lemma 3-2. We must also do a loop from R_i to O_s, O_s to O_g, and O_g to R_i. Since we picked the closest vertex to O_s in the bounding box, we can also find a minimal path from R_i to O_s to O_g to R_i. This is a shortest path that starts at a specific vertex in the bounding box, delivers the other resources, and returns to that vertex in the bounding box. O_s to O_g is a fixed distance and R_i to O_s is minimal. Once we have decided upon an R_i, O_g to R_i is also fixed. The two subproblems are solved minimally for their delivery ordering, and are independent. If a different order of delivery is shorter, we will consider that order instead. By dropping R at the boundary, the crane is carrying R for less time than if R is moved outside of the bounding box. As the crane is carrying R over fewer vertices, that means the crane is empty and available to carry a different resource over more vertices. It will never be advantageous for the crane to be full longer than it has to. Therefore, we can constrain movements and drops of R to its bounding box.

The examples in figure 3-3 are arbitrary, and the lemma holds for any two dimensional bounding box. The lemma holds if R is dropped two times at intermediate vertices. In this case, R will be dropped at R_{i1}, R_{i2}, and R_g. We can construct a similar problem where R and the crane start at R_{i1} rather than R_s. We have proven this lemma already for the single drop case, so we only have to prove that there is not an advantage to move the same resource while R is at R_{i1} and R_{i2}. That is, we want to show that the crane's actions are independent while R is at different intermediate vertices. If the crane moves resource T while R is at an intermediate vertex, there is an optimal distance that it must be moved to

T_g. Doing part of this movement while R is at R_{i1} and part at R_{i2} means that the crane must visit the same T_i twice (to drop T at T_i while R is at R_{i1}, and to pick up T at T_i while R is at R_{i2}). There is no advantage to splitting up T's delivery like this. Therefore, we can consider the crane's actions as independent when R is at different intermediate vertices, and this lemma holds for two intermediate drops. As the actions at every intermediate point are independent, this argument can be extended to any number of intermediate drops.

If we consider a one dimensional case that R_s and R_g lie along the same line in the grid, the lemma restricts R to the segment between the start and goal, and all of the same assertions still hold as the two dimensional case. □

3.2.2 Hanan grid

A Hanan grid is a graph that is constructed from a set of vertices in a plane [Hanan (1966)]. Following is a description of a Hanan grid's construction. First, the smallest bounding box that contains all of the start and goal vertices of resources is created. Vertices are added at the four corners of the box (if they are not already there) and vertical and horizontal edges are added to connect these vertices. Next, vertical and horizontal lines are drawn to intersect every start and goal vertex, and are clipped at the bounding box. Wherever the lines intersect, a new vertex is added to the graph. These line segments are added as sets of edges to the graph, where each line segment is split into edges at all the vertices it intersects. The original four bounding edges may also be replaced by multiple edges if any interior line segments intersect them. This results in a four-connected graph with edges that are either vertical or horizontal, which gives the Hanan *grid* its name. The vertices added by this construction are called Hanan points.

By its construction, the Hanan grid has an import property: a minimal rectilinear path between any pair of vertices lies on the Hanan grid. This can be easily shown by considering moving a crane horizontally and then vertically from its start to goal. Given the vertex pair (s, g), there is an unique corner point c that lies at the intersection of a horizontal line through s and a vertical line through g. Edges exist from c to both s and g, and c is a Hanan point by definition. The edges that connect (s, c) and (c, g) also exist by construction of the Hanan grid. Since the distance is measured using a rectilinear metric, these horizontal and vertical edges are the shortest distance from s to g.

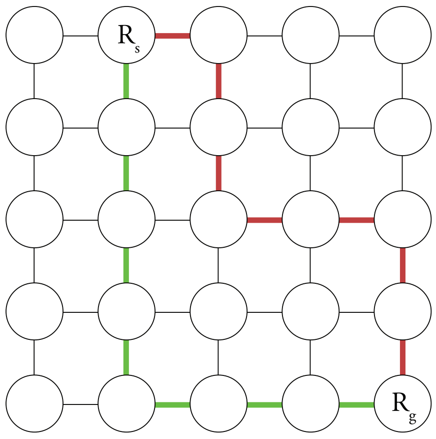
It is important to note that the size of a Hanan grid is polynomial in the number of input vertices. Given n input vertices (which corresponds to at least $\frac{n}{2}$ resources), there is a maximum of n^2 vertices in the Hanan grid. This is achievable if all of the input vertices lie along a diagonal.

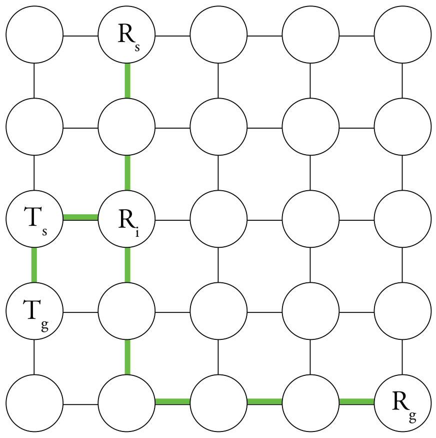
Lemma 4. *There exists an optimal solution when crane movements are restricted to a Hanan grid constructed with all resource start and goal vertices.*

Proof. Figure 3-4 illustrates an example with paths that are valid and invalid with the Hanan grid movement restriction. The red path leaves the Hanan grid constructed from $\{R_s, R_g, T_s, T_g\}$ while the green paths stay on edges contained in the Hanan grid. In Case 2, the red path is a suboptimal solution. Firstly, this lemma is trivially true when R_s and R_g lie on the same gridline. The bounding box contains a single line segment, which is also on the Hanan grid. Using lemma 3, R must move along these edges.

When R_s and R_g don't lie on the same gridline, we consider the same two cases as the previous lemma: the crane either moves R directly to R_g or drops it at an intermediate point, R_i. In case 1, R is moved to R_g with no drops. Using lemma 3-2, we can pick v to be one of the corners in the (R_s, R_g) bounding box that is not R_s or R_g. Since we picked v to be a corner point, the shortest distance from R_s to v is a straight line, and the shortest distance from v to R_g is a straight line. Both of these lines are on the Hanan grid because they are straight vertical or horizontal lines that pass through R_i or R_g.

In case 2, the intermediate drop case, R is dropped at R_i to bring other resources to their destinations. We can break this into two subproblems, similarly to case 2 of lemma 3. The crane must move from R_s to R_i to O_s to O_g to R_i to R_g. By lemma 3, R_i is the closest vertex to O_s in the bounding box of (R_i, R_g), we can show that all of the crane's movements in an optimal path lie on the Hanan grid. There are two subcases to consider: case 2a, in which O_s lies within the bounding box of (R_s, R_g), and case 2b, in which O_s lies outside





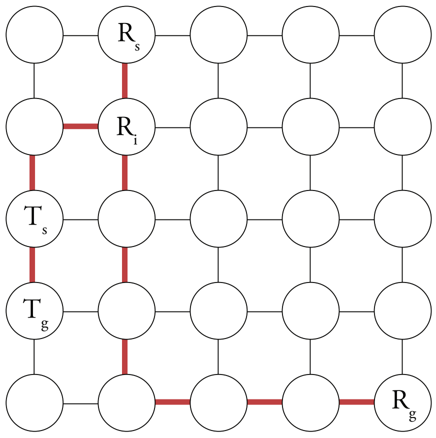


Figure 3-4: Demonstrations of lemma 4: Case 1 (top) and case 2 (center and bottom). The red path leaves the Hanan grid constructed from $\{R_s, R_g, T_s, T_g\}$ while the green paths stay on edges contained in the Hanan grid.

the bounding box. For case 2a, the algorithm in figure 3-2 can construct an optimal path from R_s to R_g that passes through O_s. All of the edges that it follows lie on a vertical or horizontal line that passes through one of the three vertices, which means that its path lies on the Hanan grid. If we set R_i to O_s, we can drop R along this path we determined to deliver R. We know that the crane can deliver the other resources between O_s and O_g along the Hanan grid by applying case 1 of this lemma. If some of the intermediate resources are also dropped, case 2 can also be applied to them. The crane can also return from O_g to O_s/R_i to pick R up again by following the Hanan grid. If we had considered a vertex besides O_s as R_i, the $R_i \to O_s \to O_g \to R_i$ loop could not be a shorter distance. Choosing a vertex closer to O_g for R_i would mean that it is farther from O_s by the same amount and no advantage is gained, and choosing a vertex farther from O_g makes both the (R_i, O_s) and the (O_g, R_i) legs longer. We have proven for case 2a that there is an optimal path if R_i equals O_s, and that this path lies on the Hanan grid.

In case 2b, when O_s is outside the bounding box, we must show that R_i is on the Hanan grid. We have already shown that picking R_i that is closest to O_s provides an optimal solution. If R_i is on a corner of the bounding box of (R_s, R_g), we know that it must be a Hanan point. If R_i is not on a corner, that means that the path from R_i to O_s is a straight line. R_i is a Hanan point because it lies on the line that passes through O_s, and is also along an edge of the bounding box of (R_s, R_g). So, the path in case 2b lies on the Hanan grid and is optimal. In all cases, an optimal solution exists if the crane is constrained to move only along the Hanan grid lines. □

Theorem 5. *An optimal solution exists when only considering drops at Hanan points.*

Proof. The intersection of Hanan grid lines are Hanan points. Lemma 4 proves that we can restrict the crane to movement along the Hanan grid without losing optimality, so it can only drop a resource on a vertex on the Hanan grid. If a resource is dropped at R_i between two adjacent Hanan points, P_i and P_j, the crane must pass through either P_i or P_j to reach R_i while staying on the Hanan grid. An example subsection of a graph with P_i

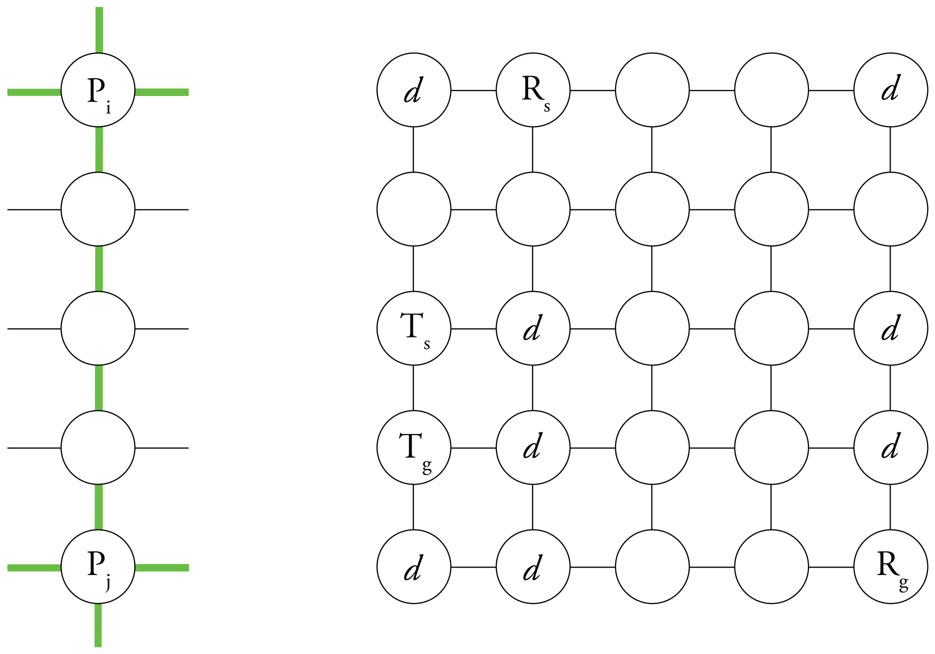


Figure 3-5: Demonstration of theorem 5: An example of two adjacent Hanan points(left), with a section of the Hanan grid edges shown in green and the original four-connected edges in black. On the right, an example with d marking all of the drop vertices that will be considered.

and P_j is shown in figure 3-5(left). The crane must later pass through either P_i or P_j in order to reach the resource at R_i and bring it to its goal. We know that the resource at R_i is not at its goal vertex because all goal vertices are Hanan points. The crane can drop the resource at P_i or P_j without adding any distance to a solution with a drop between P_i and P_j. Without loss of generality, assume that the crane will pass through P_i to pick up the resource from R_i. When the crane picks up R, it must pass through either P_i or P_j to get to R_g. In the first case, the crane moves from P_i to R_i back to P_i. This is longer than if R was already waiting at P_i. In the second case, the crane moves from P_i to R_i to P_j. This is the same length as when R_i is located at P_i, so we can ignore it as a redundant solution. Therefore, we can restrict drops to Hanan points. $\square$

Figure 3-5(right) shows all of the Hanan points on an example graph, which are the only points that the crane will consider dropping a resource. In addition to the vertices marked d, all of the other labeled vertices are also Hanan points. Note that some of the drop points are immediately rendered infeasible by lemma 3.

Lemma 6. *A path that is restricted to the Hanan grid and passes through a single Hanan point twice without passing through any other Hanan point is suboptimal.*

Proof. The four-connected graph is finer-grained than the Hanan grid, so it is possible for the crane to move along several four-connected graph edges that correspond to a single Hanan grid edge. Consider a crane path on the four-connected graph that passes through a Hanan point H twice without passing through any other Hanan point. This means that the crane didn't traverse a full Hanan grid edge, or else it would have passed through another Hanan point. By the construction of the Hanan grid, resources can only start at Hanan points. By theorem 5, resources need only be dropped at Hanan points. Therefore, there can never be a resource at a vertex that is not a Hanan point. If there are no resources lying along the edges of the Hanan grid and the crane will not consider dropping resources along those edges, there is no possible benefit to traveling part of the way between Hanan points and turning back. The only enabled actions at intermediate vertices are move actions.

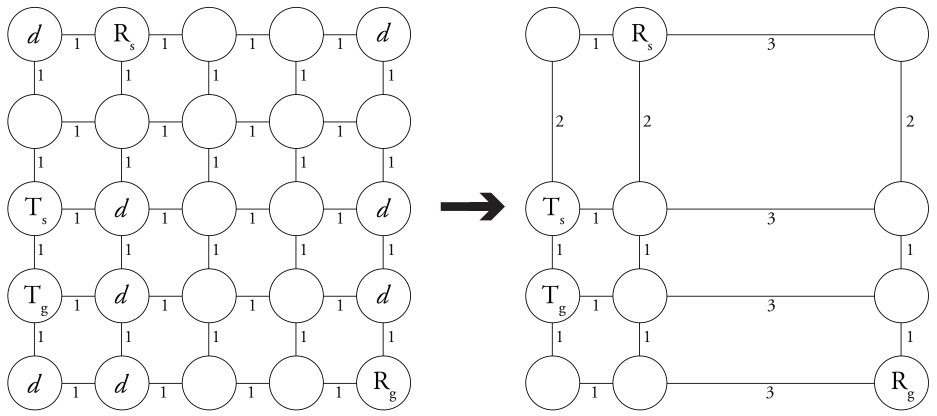


Figure 3-6: Replacement in Theorem 7: An example of a four-connected graph(left) with all of the potential drop points from Theorem 5 and the Hanan grid(right) that will be constructed from the graph.

Passing through H twice without touching another Hanan point means the system is in same state the first and second time the crane is on H. Considering this state space loop leads to a higher path cost than not considering it, so it is suboptimal and actions that lead to a loop can be ignored. As a result, there is only one valid action at each intermediate vertex: to continue moving along that Hanan edge. □

Theorem 7. *The Hanan grid can replace the four-connected graph.*

Proof. We have shown that there is an optimal solution only considering drops and, by extension, pickups at Hanan points. In addition, we showed that there is an optimal solution when only considering movements along the Hanan grid, and those movements will lead from one Hanan point to an adjacent one without turning around. This means that any vertices that are not on the Hanan grid are unreachable, and there is only one enabled action at each vertex in the four-connected graph that lies on an edge of the Hanan grid. With these restrictions, we can replace the four-connected graph with the Hanan grid that we constructed.

There is one important feature missing from the Hanan grid. The edges have no assigned costs. We can determine the cost of each edge by laying the grid over the original graph and seeing how many unit-cost edges in the original graph cover each Hanan grid edge. Assigning the sum of these four-connected edge costs to the Hanan grid edges completes the Hanan grid construction. Every action sequence on the Hanan grid can be converted in linear time to an action sequence on the four-connected graph. This is possible by scanning the action sequence over the Hanan grid and replacing each movement action by n of the same move action, where n is the cost of traversing the edge of the Hanan graph. Additionally, the cost of the sequence on the Hanan grid is equal to the cost of sequence on the original graph because of our choice in Hanan edge costs. An example of this replacement is shown in figure 3-6. Edge costs are shown are shown in both the four-connected graph and the Hanan grid. □

Lemma 8. *A Hanan grid cannot have more vertices or edges than the original four-connected graph.*

Proof. The Hanan grid is constructed based on the start and goal locations of resources. All of these locations are vertices in the four-connected graph. The Hanan points are also necessarily vertices in the four-connected graph and are therefore a subset. Every edge in the Hanan grid is comprised of at least one edge in the four-connected graph, so there cannot be more edges in the Hanan grid than the four-connected graph. Therefore, the Hanan grid is a subset of the four-connected graph. Unless resource endpoints are quite dense (at least $min(m, n)$ endpoints for an $m \times n$ graph), the Hanan grid will be a proper subset. □

Theorem 7 reduces the number of vertices on the four-connected graph in the problem. This reduction removes extraneous graph information and makes it so that the size of the graph depends only on $|A|$. For sparse graphs (graphs where most vertices are not endpoints), this can be a tremendous advantage. After applying this reduction, the problem size is independent of the original grid dimensions.

It is not guaranteed that the crane will start on the Hanan grid. Only a small modification to the Hanan grid is needed to solve a problem with this property. Adding a source

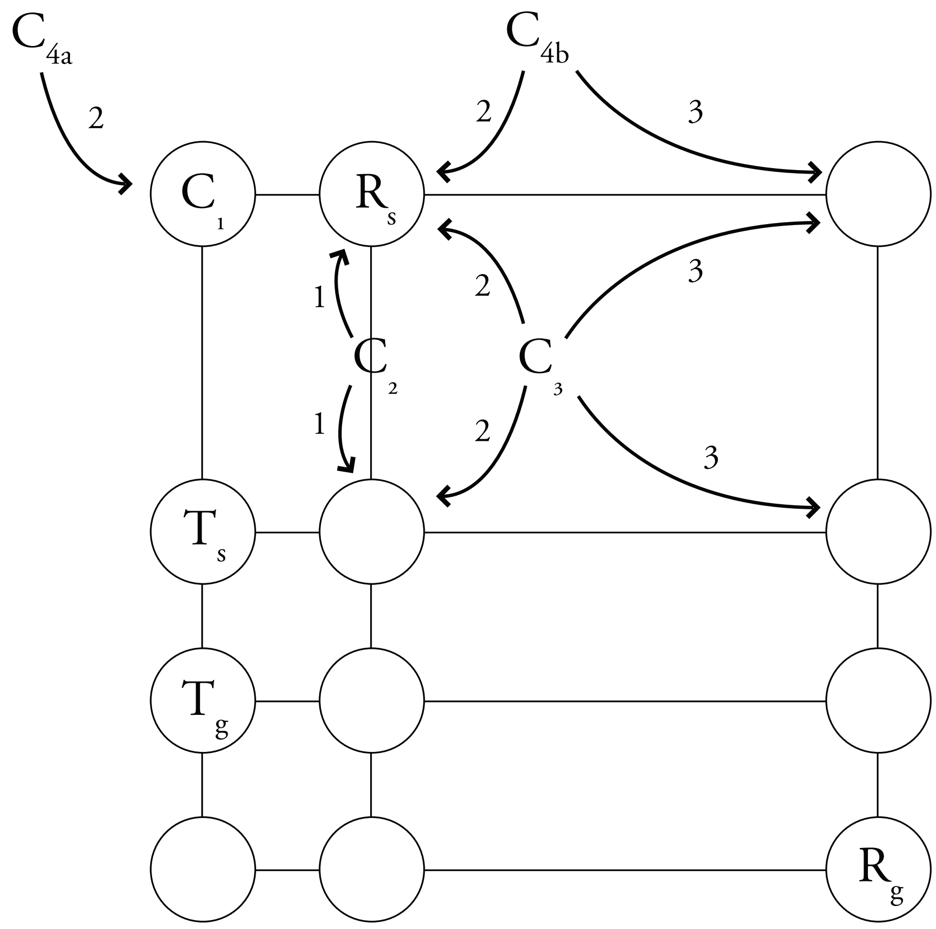


Figure 3-7: Directed edges to each case of the initial crane position. Cases 1-4 are marked with C_{1-4} and edge costs are denoted.

vertex for the crane with directed edges that lead to non-dominated Hanan points will fix this issue. A dominated Hanan point is one that is reachable optimally by passing through another Hanan point. If a solver that does not handle directed edges is being used, it is equally valid to solve several problems with the crane moved to each non-dominated vertex and using the minimum solution as a global solution.

There are four cases to consider for an initial crane position: on a Hanan point, on a Hanan grid edge, inside a Hanan grid cell, and outside of the Hanan grid. These will be cases 1, 2, 3, and 4, respectively. These cases are all demonstrated in figure 3-7. In case 1, the crane already lies on a Hanan point so there is no further work to do. In case 2, the crane lies on an edge of the Hanan grid and there are two non-dominated vertices. These vertices are the endpoints of the edge the crane starts on. In case 3, the crane lies within a grid cell and there are four non-dominated Hanan points that the crane can reach - the four corners of that grid cell. Case 4 can be split into two cases: case 4a, in which the crane is in between two Hanan points along one axis, and case 4b, in which there is one non-dominated vertex on the Hanan grid. There are two non-dominated vertices in case 4a, and only a single non-dominated vertex in case 4b. The examples of each case in figure 3-7 are generalizable. The costs of the added directed edges are determined by their rectilinear distances.

3.2.3 Corner Removal

Thus far, we have replaced the original four-connected graph with a Hanan grid that is constructed from the start and goal vertices of the resources. Because of the rectilinear metric, there are many equivalent paths in the graph. The Hanan grid can be reduced further by removing Hanan points that can be proven unnecessary. The following lemmas reduce the search space further without sacrificing optimality.

Definition 2. Corner vertex

A corner vertex is any vertex that has exactly one horizontal edge and one vertical edge.

Definition 3. Diagonally opposite vertex

```
removeCorners($V, E, A$)
 1. $Q \leftarrow$ corner vertices of $V$
 2. while $Q$
 3.     $v \leftarrow Q.pop$
 4.     if $v \in$ pair in $A$ then continue
 5.     if $v'$ does not exist then continue
 6.     $V \leftarrow V \backslash \{v\}$
 7.     $E \leftarrow E \backslash \{(v, *), (*, v)\}$
 8.     foreach $n \in v.neighbors$
 9.         if $n$ is a corner vertex
10.             $Q.push(n)$
11. Return $V, E$
```

Figure 3-8: The corner-removal algorithm.

Every corner vertex v has a unique location where a diagonally opposite vertex may be. This location is found by adding the two edges of v in a vector fashion. If a vertex exists at this location, it is considered diagonally opposite v and is termed v'.

Theorem 9. *The algorithm described in figure 3-8 maintains an optimal solution when the original graph is a four-connected grid.*

Proof. Because of symmetries, we can remove certain Hanan points and their edges using the algorithm in figure 3-8, resulting in a reduced Hanan grid. Since the graph starts as a grid, every pair of vertices is connected by edges with minimal rectilinear distance. We are only interested in preserving minimal paths between start and goal vertices, which will be referred to as 'important' vertices. Only corner vertices are removed, so vertical and horizontal convexity is preserved. That is, at least one minimal path remains for important every pair of vertices. We can prove this for both one and two dimensional cases.

In the one dimensional case, the entire Hanan grid lies along a single line. In this case, there are either no vertical edges or no horizontal edges. That means that there are no corner vertices, which means no vertices are removed and the graph is unchanged.

The two dimensional case will be proven by induction. The original graph, the Hanan grid, has an optimal path between every pair of vertices. Given a graph that has optimal paths between every important vertex pair, consider executing lines 2-10 once on vertex c. c has no more than two edges because only corners are added to Q. It has no fewer than one edge because a vertex with zero edges would mean that the graph is disjoint, and the algorithm cannot remove a vertex that causes the graph to become disjoint. There are two cases to consider: c is not removed and c is removed. If c is not removed, the graph is unchanged and therefore still has optimal paths between every pair of important vertices. If c is removed, we are guaranteed that the graph is still connected because c' exists. We also know that c is not an important vertex. We can show that a path with equal distance exists for any optimal path that passes through c. If c has one edge, there are no optimal paths between important vertices that pass through it. Otherwise, all optimal paths that pass through c can be routed through c' instead. Passing through c' instead of c only means that vertical and horizontal movements are swapped, so the path length remains the same. Examples of this case are demonstrated in figure 3-9.

It follows from this that a shortest path between every important pair of vertices is preserved, and an optimal solution exists within the graph reduced by the algorithm in figure 3-8. □

Lemma 10. *The algorithm described in figure 3-8 cannot remove vertices on a resource's bounding box that are closest to the start location of a resource that lies outside the bounding box.*

Proof. This lemma has a few cases to handle. First, there is only one way that a corner of the initial Hanan grid can have this property. That is if a corner lies within a one dimensional bounding box. In this case, the corner must also be an important vertex, as it only has an edge on one side of it. Since it is important, it will never be removed. Besides the case of an important corner vertex on a one dimensional bounding box, it is not possible for a vertex on a bounding box that is closest to another resource to become a corner. This can be proven by contradiction. Assume vertex v is on the bounding box of (R_s, R_g) and is closest to T_s and is a corner. We know that v is both a corner and on a bounding box. That means that v's edges lead to vertices that are also on the bounding box. T_s lies outside of the bounding box, however, and there must be a path from v to T_s. The only possible paths pass through one of v's neighboring vertices, which means that one of these vertices is closer to T_s than v is. Therefore, v is not the closest vertex to T_s and we have a contradiction. Because of this contradiction, we know that if a vertex is both on a bounding box and closest to another start vertex, then the vertex cannot become a corner. This, in turn, means that the vertex cannot be added to Q to be removed. □

Figure 3-9 shows an example run of the algorithm in figure 3-8 on the example we have been using. 3-9a shows the input: vertices, edges, and pairs of start/goal vertices. 3-9b-3-9k show the algorithm during execution, paused after line 3 executes. Line 3-9l shows the graph that the corner-removal algorithm will return. In figure 3-9b, the starting corners are added to Q and the top left corner has just been popped into v. In figure 3-9b, figure 3-9c, and figure 3-9d, corners are popped from Q and removed from V, and their adjacent vertices are added to Q. In figure 3-9e, $v \in A$, so the vertex is not removed from V. In figure 3-9h, only one of the adjacent vertices is added to Q because the other one is not a corner vertex. Removing v in figure 3-9k would cause V to become disjoint, so the vertex is not removed. The final graph still has optimal paths between all pairs of important vertices after redundant corners were removed.

In addition to theorem 9 maintaining shortest paths between pairs of important vertices, we can show that it runs in $O(|V| * |A|)$ time. The loop in line 2 runs at most $|V|$ times. Line 4 may run as many as $|A|$ times. Every other operation inside the while loop runs in constant time. This may be surprising, as determining whether a graph is disjoint usually takes more computation. In this case, however, we can take advantage of an important property mentioned in theorem 9. Corner c is only removed if its diagonal vertex, c' exists. This means that checking if the graph would be disjoint without c is a single operation.

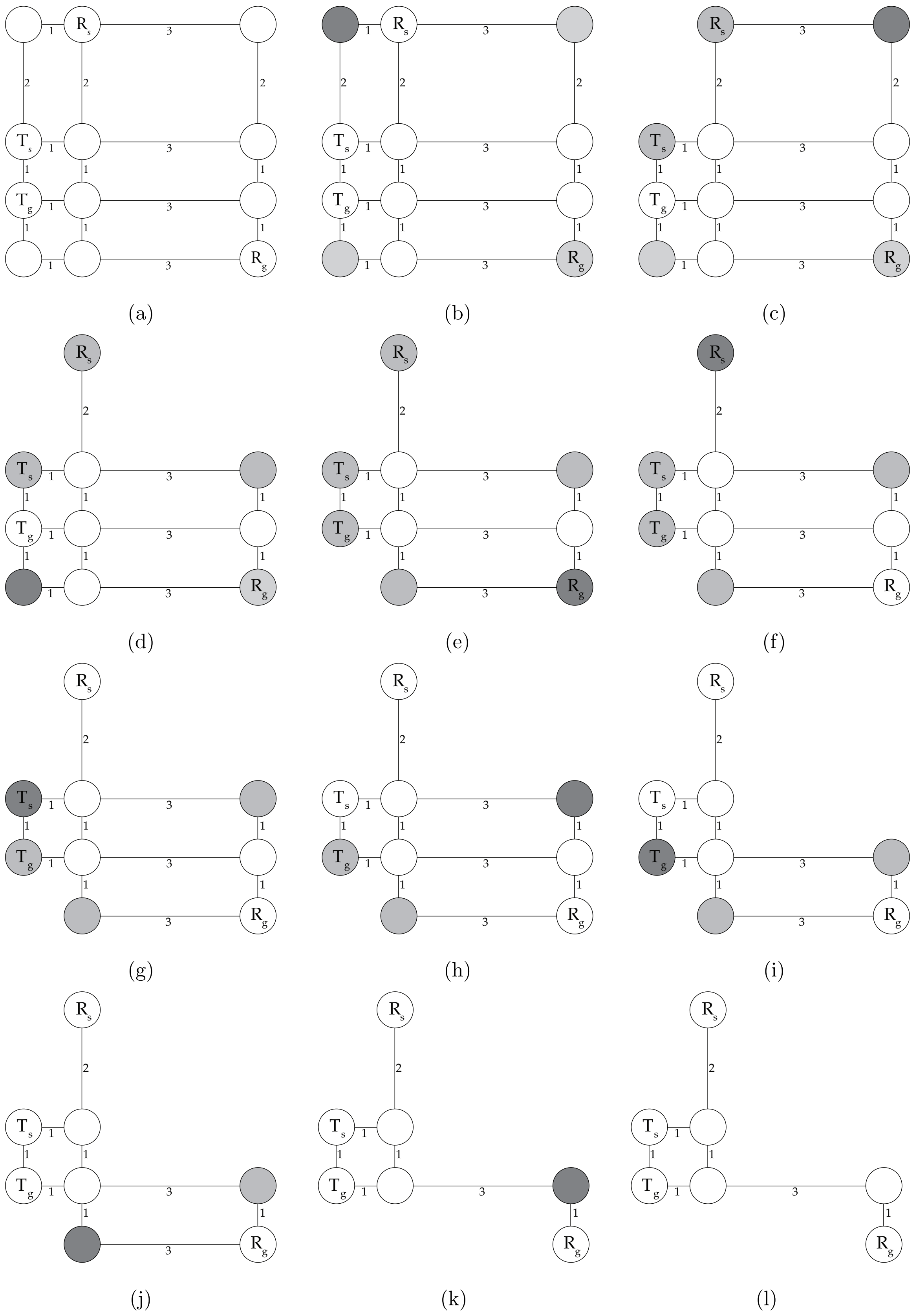


Figure 3-9: Example of algorithm in figure 3-8 running. Light grey vertices are in Q and the dark grey vertex is the current v being considered.

Each corner has, by definition, two edges. On the four-connected grid, each neighbor vertex has a maximum of four edges, so line 8 involves at most 8 edge checks. As one loop is nested in another, we multiply their number of operations and arrive at $O(|V| * |A|)$ running time for the algorithm in figure 3-8.

CHAPTER 4

Discussion

This chapter provides experimental results of graph reductions on randomly generated problems, for both the Hanan grid reduction and the corner removal reduction. It also discusses some important limitations to the ideas in this thesis as well as possible extensions to the work.

4.1 Experiments

Using the theorems presented in this paper, we can compare the graph sizes of the original state space versus the reduced state space. The experiments section discusses an approach to generating SCP problems, reducing them, and comparisons of graph sizes.

4.1.1 Instance Generation

There is not a large corpus of real-world SCP instances[Cirasella et al. (2001)]. Therefore, we need a way to generate random tests. The algorithm presented in Cirasella's paper was used as a reference and modified it for rectilinear space, as shown in the algorithm in Figure 4-1. The instance generator takes three parameters: the bounds of the graph, the number of resources to move, and u. u dictates the maximum arc distance allowed, which can strongly influence the structure of the problem. When u is small, a small percentage of the total distance that the crane travels is along the arcs. Most of the distance that the crane travels is travelled without carrying a resource. When u is large, more of the total distance that the crane travels is traveled while carrying a resource.

The algorithm works as follows: A starting location for each resource is chosen randomly. An offset for each resource (bounded by u) is chosen randomly and checked for bounds. This

```
generateInstance($u, bounds, numRes$)
1. $A \leftarrow \{\}$
2. for $i \leftarrow 0..numRes$
3.     $s \leftarrow (rand(), rand())$
4.     do $offset \leftarrow (rand()/u, rand()/u)$
5.         while $s + offset \not\subset bounds$
6.     $A \leftarrow A \cup \{(s, s + offset)\}$
7. $C_i \leftarrow (rand(), rand())$
8. Return $A, C_i$
```

Figure 4-1: Algorithm to generate random SCPs.

resulting offset gives the destination of the resource. A random starting location for the crane is chosen. Finally, (source, destination) pairs of each resource are returned, along with the initial crane location.

The input parameters allow for very different types of problems to be generated. As u decreases and the bounds become very large, the resulting SCPs are approximable with TSPs. We can see this by observing the total distance the resources must be carried, $d_{tot} = \sum_i |R_{is} - R_{ig}|$, in comparison to the total distance the crane must go, C_{tot}, becomes negligible. If $u = 1$ and the bounds are 10000×10000, a two-resource problem could end up with C_{tot} being thousands of times larger than d_{tot}. If $u = 1$, in fact, there will never be any intermediate drops because the bounding box for each resource only allows the resource to be moved from its start to its goal. This case is almost exactly the same as the Traveling Salesman Problem because the solution only involves finding which order to visit resources.

4.1.2 Experimental Results

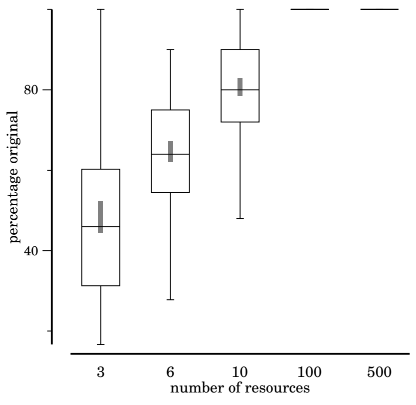
Using randomly generated instances with varied parameters, we can see the impact on vertex counts that the reductions in this thesis have. The benefits from reductions diminish, predictably, as the ratio of resources:vertices increases, as that means the graph is getting denser and the reduction is less likely to remove vertices. Graphs were generated for $bounds \times bounds$ squares using $u < bounds$. Generating graphs with $u > bounds$ would be redundant, as the distance between each start and goal vertex is limited by the grid size.

Each plot in figure 4-2 demonstrates the average percentage reduction on 100 plots that were randomly generated using the algorithm in figure 4-1. In the 10×10 instances, a value of 4 was used for u. In the 100×100 instances, 60 was used. In all other instances, a value of 100 was used. The plots show the impact of the Hanan grid reduction. The y axis shows the percentage of vertices in the original four-connected grid. The box in each plot surrounds the middle half of the results. The horizontal line in the box represents the median value and the whiskers extend to the min and max. Circles beyond the whiskers are considered outliers. The caption of each graph is the original grid size, though each graph was truncated to its resource's bounding box. For example, a generated 10×10 instance may not have any important vertices in the left two columns. The original grid is considered to be 8×10 in this case, which decreases the impact of the Hanan grid reduction.

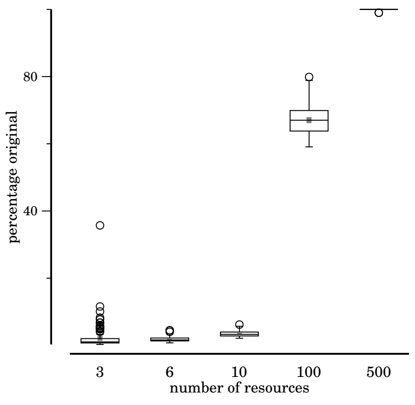
For larger grid sizes, the results show that the Hanan grid size depends almost completely on the number of resources rather than the size of the grid. As the density of resources on the grid increases, we see both smaller reductions and more variations in reduction size. Figure 4-2a shows an average reduction of over 50% for the three resource case, while the board is so dense in the 500 resource case that no reductions are possible.

The plots in figure 4-3 show the reduction from the Hanan grid to the Hanan grid with corner removal applied. Each plot is marked with the original size of the four-connected grid, even though the plots use the number of vertices on the Hanan grid as a baseline. Corner removal works very well for problems with few resources. The graph is reduced by at least 40% in the 3 resource case. Even for larger grids, we see the reduced graph has only 60-70% of the original vertices for most of the 10 resource cases. Rather than the Hanan grid reduction that does not reduce the grid at all when resources are dense, corner removal still deletes some vertices in all but the most extreme case (500 resources on a 10×10 grid).

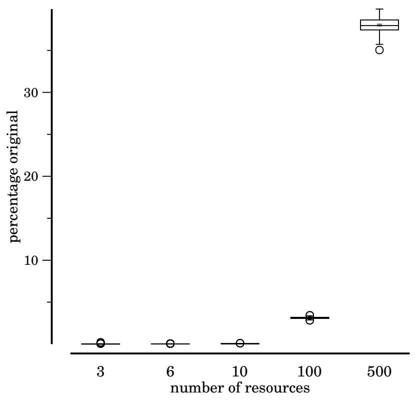
These plots support two ideas that were mentioned earlier. First, the density of the



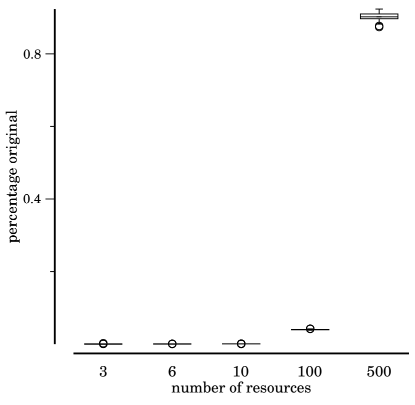
(a) 10 × 10



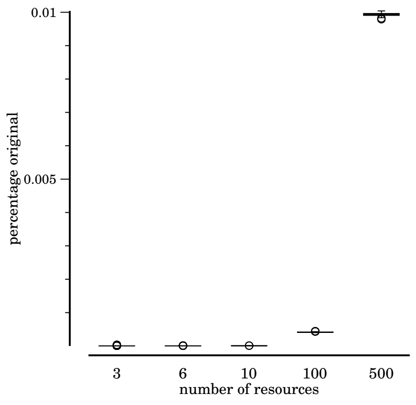
(b) 100 × 100



(c) 1000 × 1000

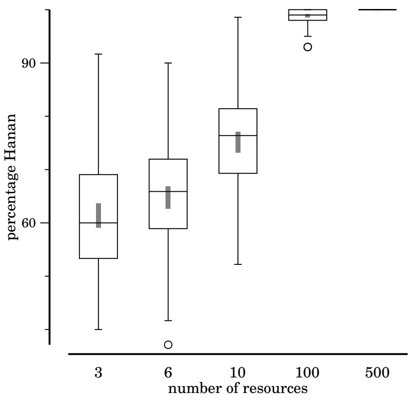


(d) 10000 × 10000

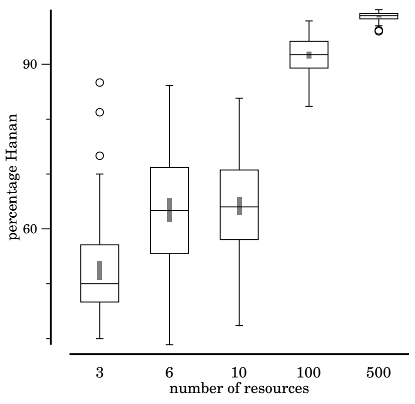


(e) 100000 × 100000

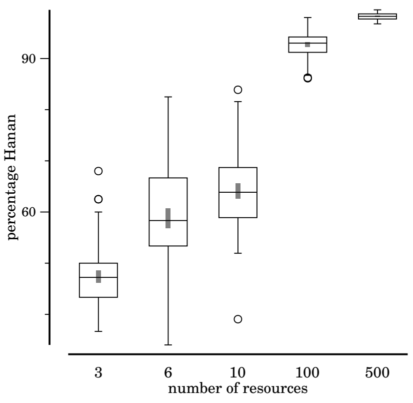
Figure 4-2: Vertex reduction percentages from a four-connected grid to Hanan grid.



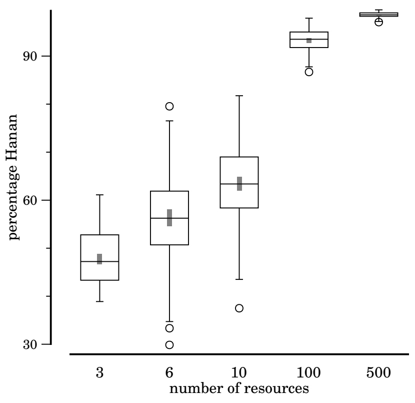
(a) 10×10



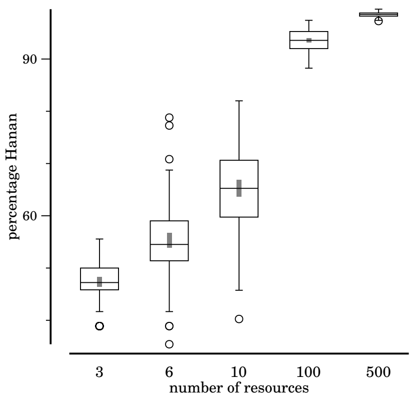
(b) 100×100



(c) 1000×1000



(d) 10000×10000



(e) 100000×100000

Figure 4-3: Vertex reduction percentages by applying the corner removal algorithm to a Hanan grid.

resources on the grid has a major impact on both reductions. A denser graph means that there are fewer vertices that are not important, which provides fewer chances to remove a vertex. Second, these reductions can have a real and noticeable impact on the size of the state space. If both the Hanan grid reduction and corner removal have 50% reductions, the final graph size will be 75% smaller.

4.2 SCP in Higher Dimensions

All of the work presented here is defined for a two dimensional SCP. (One dimensional SCPs can be solved in polynomial time, so reducing their state space is not as interesting.) Because distances are measured using Manhattan distance, there are no assumptions made that only apply to two dimensions. An n-dimensional SCP problem could be reduced using the theorems in this thesis with only a few small definitional changes. Rather than a two dimensional bounding box with corners on the start and destination of a resource, the bounding box would be defined as an n-dimensional box that otherwise has the same properties. The box would have faces instead of edges, but it is still possible to find minimal paths as the distance in each direction is independent. A Hanan grid would be constructed by extending edges and creating vertices in every direction rather than just two.

To apply corner removal, corners would have to be redefined, as one edge in each of two directions would no longer apply in n dimensions. Instead a corner would be defined as a vertex that has one edge for each of n different directions. Also, a diagonally opposite vertex would be determined by vector-wise summing each of the n edges on the corner. For k resources, a Hanan grid may have as many as k^n vertices, which becomes very large very quickly.

4.3 Limitations and Extensions

There are many limitations to this reduction algorithm. Some of the limitations come from the chosen topology and some from other problem and solution properties. The integral choice of using a four-connected graph may be useful in some situations, but a poor approximation in others. If an agent can move diagonally, restricting it to a rectilinear grid increases its pathlength unnecessarily. An additional environmental hazard that we may encounter is an obstacle in the grid. This reduction will not provide correct results if the initial four-connected graph has ‘holes’ in it, which may represent obstacles that interfere with the agent. Problems that are more dense, those that have a large ratio of resources to vertices, derive little benefit from the reductions. Other search techniques may better solve dense graphs like these.

4.3.1 Euclidean Distance

Many of the theorems in this thesis rely on properties that rectilinear measurements exhibit, but Euclidean measurements do not. If an agent can move in arbitrary directions, restricting it to a grid will negatively impact its overall pathlength. In a worst case scenario, the rectilinear distance of a path may be up to $\frac{2}{\sqrt{2}} \approx 1.41$ times longer than the Euclidean distance. The rectilinear distance can never be shorter than the Euclidean distance. This is a loss of nearly the same efficiency that we gained by allowing for preemption. If we want to model a preemptive SCP using Euclidean distance, we will have to approach it from a very different way.

The four-connected graph construction that we have used does not represent possible motion transitions in this situation. If we choose to discretize the movement space, we will have to work with a fully connected graph as the agent may move at any angle. Rather than approaching the problem by removing vertices as we did in the rectilinear case, we may start with a fully connected graph on the vertices in A and add as few vertices as needed to guarantee an optimal solution. In the non-preemptive case, this is simply the original SCP. Since preemption is allowed, we have to pick some additional vertices that would allow for an optimal solution. We can certainly consider a theorem similar to lemma 1. In this case, we are using Euclidean distance and can confine drop points to the convex hull of A.

4.3.2 Obstacles

The reduction algorithms assume that the initial graph has no holes – the Hanan grid reduction relies on this for combining edges and the corner removal algorithm relies on this for initializing its queue. The corner removal algorithm will preserve all shortest paths between important vertices if its initialization step is modified to add any interior corners. The Hanan grid reduction, however, combines and deletes edges and may remove optimal all optimal paths between two important vertices.

In physical domains, obstacles can be an important feature to model. There may be objects on the floor to avoid, or separate rooms with narrow halls or doorways connecting them. In these cases, the whole search space cannot be reduced with the same approach as we took. It would still be possible to reduce subsections of the grid, but it would require much more processing to find valid subsections to reduce. Harabor's work is better adapted to pathfinding with separate rooms, though neither his work nor this one deals well with smaller obstacles.

4.3.3 Steiner Tree Guidance

Steiner minimal trees are trees which connect a set of vertices V with a minimum length of line segments [Gilbert and Pollak (1968)]. Rather than minimal spanning trees which only allow original vertices to be used, Steiner trees allow the introduction of new vertices if these vertices reduce the total length of the edges connecting V. Finding the Steiner minimal tree of a set of vertices is NP-complete for both Euclidean and rectilinear space [Karp (1972), Garey and Johnson (1977)]. Steiner points have the same goal as drop points in the preemptive SCP: to reduce the total distance between more than two points.

It may be possible to constructively build a graph from A and add the set of Steiner points as additional drop points to still achieve an optimal pathlength. In an algorithm like A*, the Steiner points would have to be generated for each subgraph when considering states that have some delivered resources.

4.3.4 Further Reductions

It is evident that there are still symmetrical paths, meaning that there are still redundant vertices. The simple example that we have used in this thesis (in figure 3-9l) shows that at least two further reductions are possible. The vertex immediately above R_g is not an important vertex and only has two edges. Like in lemma 6, there is never a situation where changing directions or dropping a resource at that vertex will be advantageous. It can be removed and its edge costs summed to create a singe new edge.

In addition to this reduction, the vertex to the right of T_g is not necessary. Some of the lemmas only consider dropping resource R at the vertex in the bounding box of (R_s, R_g) that is closest to another resource's start vertex. It is evident that the vertex to the right of T_g is not a vertex that is closest to any start vertex, so it may be removed from consideration as a drop point. It has up to four edges, so up to six edges may replace it (so that all of its neighbor vertices are connected to each other). This reduction allows for removing interior vertices, which other reductions have not.

4.3.5 Optimality

While all of the reductions in this thesis aim at preserving optimal solutions, there is no requirement that an optimal solver be used. As mentioned in chapter 2, bounded suboptimal approximation algorithms exist for other types of SCPs. Although the solutions are only approximate, these algorithms have the advantage of running in polynomial time. Much larger problems can be solved approximately. It is probably possible to adapt approximation algorithms from the general SCP or related problems to the four-connected SCP. It is also possible to use non-optimal heuristic search solvers to find solutions quickly. For example, a greedy solver that always considers the action that appears best could provide a feasible solution very quickly. Because a solver has fewer vertices and fewer actions to consider, the reduction presented in this thesis would speed many of these approximation methods as well as optimal ones.

CHAPTER 5

Conclusions

This thesis has explored graph reductions of a four-connected grid for preemptive stacker-crane problems. Two graph reductions were proposed, and both were proven to maintain an optimal solution. The first reduction replaced the four-connected grid with a Hanan grid constructed from the endpoints of A. It was proven that this Hanan grid maintains optimal solutions by keeping shortest paths between all important vertices. The reduction makes graph size and four-connected grid size independent, as the Hanan grid depends only on the number of endpoints. For large graphs with a small number of resources, this may remove most of the vertices in the graph, allowing for faster analysis.

The second reduction removes extraneous corners from the graph. It is proven that the algorithm presented for this reduction also maintains shortest paths between important vertices. Corners are evaluated for redundancy and if they serve no unique purpose, they are removed from the graph. This analysis is very quick and, as we saw in the presented example graph, can sometimes remove a significant number of vertices. The experimental results shown in the discussion section demonstrate that the reductions are effective when resources are not overly dense on the graph.

These reductions can be used in conjunction with any graph solver. They preserve optimal solutions so an optimal solver will find correct results from the reduced graph. An approximate solver can also be used, and may run faster as there are fewer vertices on the graph. The graph size is the mantissa of the state space size, so reducing it will reduce the state space significantly.

BIBLIOGRAPHY

Shoshana Anily, Michel Gendreau, Gilbert Laporte, and Québec) Groupe d'études et de recherche en analyse des décisions (Montréal. *The preemptive swapping problem on a tree*. Montréal: Groupe d'études et de recherche en analyse des décisions, 2006.

Mikhail J Atallah and S Rao Kosaraju. Efficient solutions to some transportation problems with applications to minimizing robot arm travel. *SIAM Journal on Computing*, 17(5): 849–869, 1988.

Michael O Ball and Michael J Magazine. Sequencing of insertions in printed circuit board assembly. *Operations Research*, pages 192–201, 1988.

Gerardo Berbeglia, Jean-François Cordeau, Irina Gribkovskaia, and Gilbert Laporte. Static pickup and delivery problems: a classification scheme and survey. *Top*, 15(1):1–31, 2007.

Jill Cirasella, David Johnson, Lyle McGeoch, and Weixiong Zhang. The asymmetric traveling salesman problem: Algorithms, instance generators, and tests. *Algorithm Engineering and Experimentation*, pages 32–59, 2001.

G.N. Frederickson, M.S. Hecht, and C.E. Kim. Approximation algorithms for some routing problems. In *17th Annual Symposium on Foundations of Computer Science, 1976*, pages 216–227. IEEE, 1976.

Michael R Garey and David S. Johnson. The rectilinear steiner tree problem is NP-complete. *SIAM Journal on Applied Mathematics*, 32(4):826–834, 1977.

E.N. Gilbert and H.O. Pollak. Steiner minimal trees. *SIAM Journal on Applied Mathematics*, 16(1):1–29, 1968.

Maurice Hanan. On steiner's problem with rectilinear distance. *SIAM Journal on Applied Mathematics*, 14(2):255–265, 1966.

Daniel Harabor and Adi Botea. Breaking path symmetries in 4-connected grid maps. *AIIDE-10*, pages 33–38, 2010.

Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2):100–107, 1968.

Richard M Karp. *Reducibility among combinatorial problems*. Springer, 1972.

Hervé LM Kerivin, Mathieu Lacroix, and Ali Ridha Mahjoub. Models for the single-vehicle preemptive pickup and delivery problem. *Journal of combinatorial optimization*, 23(2): 196–223, 2012.