ABSTRACT

Title of Document:	MARITIME PIRACY: SOLVING THE OPTIMIZED TRANSIT PATH PROBLEM
	John Robert Schedel, Jr., Ph.D., 2015
Directed By:	Professor Gregory B. Baecher
	Department of Civil and Environmental Engineering

Models have been developed that accurately predict the probability of pirate activity at locations throughout the Arabian Sea. With these piracy prediction models, mariners transiting this region can ensure that their course avoids the highest threat regions and that ample anti-piracy precautions are in place elsewhere. However, they are on their own to determine their "best" transit path.

Using unique piracy success predictors and transit cost calculators, along with existing pirate activity predictions, this research develops a method for determining the Optimized Transit Path through the Arabian Sea. This method simultaneously optimizes two different attributes, piracy avoidance and cost minimization, based on a mariner's priorities.

The Optimized Transit Path (OTP) algorithm calculates the minimum cumulative path through a two-dimensional, geographic matrix. The OTP algorithm finds the shortest path through the network from a starting line on one side of the matrix to a finish line on the other side. Using a computer code of the algorithm, experimental tests quantified the OTP algorithm's computation

speed and required number of calculations to reach a solution. Further, the performance of the OTP algorithm at solving the piracy matrix was compared to the speed of other shortest path algorithms. Based on this study, the OTP algorithm's speed at solving the piracy matrix was comparable to that of the fastest shortest path algorithm in use today, Dijkstra's Algorithm implementing a Min-Priority queue with a Fibonacci Heap, and significantly faster than all others.

Because it can use the piracy prediction matrix directly as an input, the OTP algorithm is especially well suited for solving the piracy avoidance problem. More importantly, its calculation of Optimized Slack quantifies the additional cost of diverting from the shortest path, information not calculated by other shortest path methods. However, use of the OTP algorithm is fairly limited, as it is only well suited for matrices that represent a flat plane of interconnected geographic areas, with movement from a node limited to the eight adjacent nodes surrounding it.

Another promising application of the methods in this paper is within the field of underwater search.

Models have been developed that accurately predict the probability of pirate activity at locations throughout the Arabian Sea. With these piracy prediction models, mariners transiting this region can ensure that their course avoids the highest threat regions and that ample anti-piracy precautions are in place elsewhere. However, they are on their own to determine their "best" transit path.

Using unique piracy success predictors and transit cost calculators, along with existing pirate activity predictions, this research develops a method for determining the Optimized Transit Path through the Arabian Sea. This method simultaneously optimizes two different attributes, piracy avoidance and cost minimization, based on a mariner's priorities.

The Optimized Transit Path (OTP) algorithm calculates the minimum cumulative path through a two-dimensional, geographic matrix. The OTP algorithm finds the shortest path through the network from a starting line on one side of the matrix to a finish line on the other side. Using a computer code of the algorithm, experimental tests quantified the OTP algorithm's computation speed and required number of calculations to reach a solution. Further, the performance of the OTP algorithm at solving the piracy matrix was compared to the speed of other shortest path algorithms. Based on this study, the OTP algorithm's speed at solving the piracy matrix was comparable to that of the fastest shortest path algorithm in use today, Dijkstra's Algorithm implementing a Min-Priority queue with a Fibonacci Heap, and significantly faster than all others.

Because it can use the piracy prediction matrix directly as an input, the OTP algorithm is especially well suited for solving the piracy avoidance problem. More importantly, its calculation of Optimized Slack quantifies the additional cost of diverting from the shortest path, information not calculated by other shortest path methods. However, use of the OTP algorithm is fairly limited, as it is only well suited for matrices that represent a flat plane of interconnected geographic areas, with movement from a node limited to the eight adjacent nodes surrounding it.

Another promising application of the methods in this paper is within the field of underwater search.

MARITIME PIRACY: SOLVING THE OPTIMIZED TRANSIT PATH PROBLEM

By

John Robert Schedel, Jr.

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park, in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2015

Table of Contents

List of Tables

List of Figures

List of Abbreviations

CASP – Computer Aided Search Program

CPM – Critical Path Method

EMV – Expected Monetary Value

MF – Minimum Finish of a node

MS – Minimum Start of a node

OS – Optimized Slack of a node

OTP – Optimized Transit Path

PARS – Pirate Attack Risk Surface

PPS – Piracy Prediction Surface

RF – Required Finish of a node

RS – Required Start of a node

RTM – Risk Terrain Modeling

UKGM – United Kingdom Global Model

Chapter 1: Introduction and Background on Maritime Piracy

1.1 Maritime Piracy: "A Global Phenomenon. A Local Problem."

For most people, the word piracy invokes romanticized images of a past age, of sailing ships and walking the plank. Blackbeard might come to mind or, more likely, Johnny Depp's character Captain Jack Sparrow from the "*Pirates of the Caribbean*" movies. Few people realize that piracy on the high seas is alive and well in the world today. In fact, maritime piracy has undergone a major revival over the past two decades and is now a multi-billion dollar per year enterprise. The 2013 Hollywood blockbuster, "*Captain Phillips*," paints a fairly accurate picture of modern pirates, who are far more brutal, desperate, and high-tech than their predecessors of an earlier age.

The International Maritime Bureau (IMB), the world's foremost authority for monitoring pirate activity, defines maritime piracy as "an act of boarding or attempting to board any ship with the intent to commit theft or any other crime and with the intent or capability to use force in the furtherance of that act" (ICC-IMB 2014). Though different countries and legal authorities sometimes disagree on geographical jurisdictions and definitions related to maritime piracy, all seem to agree on the three main tenets of the IMB definition: intent to board, intent to commit a crime and intent (or capability) to use force.

In today's world, maritime piracy can best be described as a "global phenomenon but a local problem" (Murphy 2010). Piracy is prevalent in localized regions around the globe where geography, economics and politics converge to make it profitable and feasible. In each region, the tactics and objectives of piracy differ, making a "one size fits all" solution impractical. Figure 1 shows the global areas of significant pirate activity (shaded areas) over the past decade.

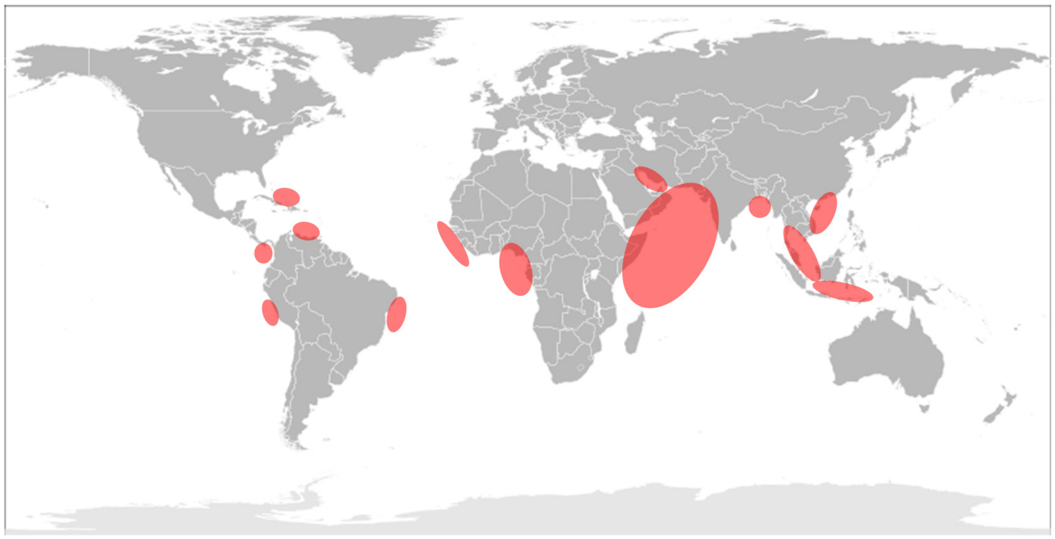


Figure 1: Global Maritime Piracy Activity
(Source: Map from Wikimedia Commons. Data adapted from Galletti 2007 and ICC-IMB 2014.)

Globally, there are eight geographic "hot spots" for maritime piracy (Scheffler 2010) :

- Arabian Sea (Gulf of Aden / Gulf of Oman / Somalia)
- West Indian Ocean (from Somalia south to Seychelles)
- Gulf of Guinea (West Africa off Nigeria and Benin)
- Straits of Malacca (Indonesia / Malaysia / Singapore)
- South China Sea (Thailand / Malaysia)
- Celebes Sea (Philippines / Indonesia)
- Bay of Bengal (India / Myanmar)
- South America

Piracy tactics and objectives differ within these regions. In some regions, pirates intend solely to rob the ships then depart. Attacks usually occur clandestinely at night, often to ships at anchor. Anything easily transported, hard to track and quick to convert to cash is targeted, from cash in the captain's safe to paint in the ship's lockers. Ships at anchor are frequently targeted, and thefts are sometimes not discovered until the next day (Dillon 2005).

In other regions, pirates intend to hijack the ship, sail it to a safe location then sell the cargo and/or the entire ship (once it has been properly renamed and renumbered). These hijackings with the intent to sell tend to be the most violent of pirate activities, as surviving crew members could reveal the true origins of the ship and cargo (Chalk 2009).

Another piracy objective, prevalent in the 1990's and early 2000's but now mostly absent, is to hijack the ship, sell its cargo, then use it as a "phantom ship." Pirates disguise a "phantom ship" as a legitimate cargo ship, contract to carry cargo in a busy port and load the ship. Upon leaving port, the "phantom ship" simply disappears with the cargo, which the pirates sell in another port after renaming and renumbering the ship (Chalk 2008).

The final tactic, which is used especially in Somalia and Nigeria, is for the pirates to hijack the ship then hold the ship, cargo and crew for ransom. Once the shipping company or their insurers pay a ransom, often in the millions of dollars, the ship, cargo, and crew are returned to them. This hijack and ransom tactic has proven the most profitable and disruptive of all the piracy methods (Nincic 2009).

Globally, the number of annual piracy attacks saw a steady increase throughout the 1990's then a leveling off to about 300 to 400 incidents per year in the 21st century. Table 1 shows the annual number of piracy attacks, sorted by geographic region, from 1991 to 2013 (Geopolicity 2011 and ICC-IMB 2014). Though the worldwide number of attacks has held fairly steady for the past 15 years, the geographic concentration of these attacks has changed. In the late 1990's, an explosion of piracy in Southeast Asia accounted for an increase in annual numbers of attacks. However, the Indian Ocean tsunami of 2005 destroyed land support bases for Southeast Asian piracy, dramatically reducing its prevalence (Jones 2011). At about the same time, maritime piracy in the Arabian Sea, based out of Somalia, escalated and is now the leading location for pirate activity, both in number of incidents and in economic impact.

Table 1: Annual Number of Piracy Attacks from 1991 to 2013
Source: Data from Geopolicity 2011 and ICC-IMB 2014

Global Piracy Incidents (1991-2013) by Region

	1991	1992	1993	1994	1995	1996	1997	1998	1999	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013
South East Asia	88	63	16	38	71	124	92	89	161	242	153	153	170	158	102	83	70	54	46	70	80	104	128
Africa	0	0	7	6	20	25	46	41	55	68	85	78	93	73	80	61	123	189	270	259	293	150	79
Indian Sub-Continent	0	5	3	3	16	24	37	22	45	93	53	52	87	32	36	53	30	23	30	28	16	19	26
America	0	0	6	11	21	32	37	35	28	39	21	65	72	45	25	29	21	14	37	40	25	17	18
Far East	14	7	69	32	47	17	19	10	6	20	17	17	19	15	20	5	10	11	23	44	23	7	13
Rest of World	5	31	2	0	13	6	17	5	5	7	6	5	4	6	13	8	9	2	4	4	2	0	0
TOTAL	107	106	103	90	188	228	248	202	300	469	335	370	445	329	276	239	263	293	410	445	439	297	264

Economically, the international cost of piracy is significant and expected to grow. In 2010, the annual cost of piracy was estimated at between \$5 and \$8 billion. By 2015, the annual cost is expected to grow to \$13 to \$15 billion (Geopolicity 2011).

Most of this rise in cost is tied to the shifting of piracy objectives to the hijack and ransom model practiced in and around Somalia.

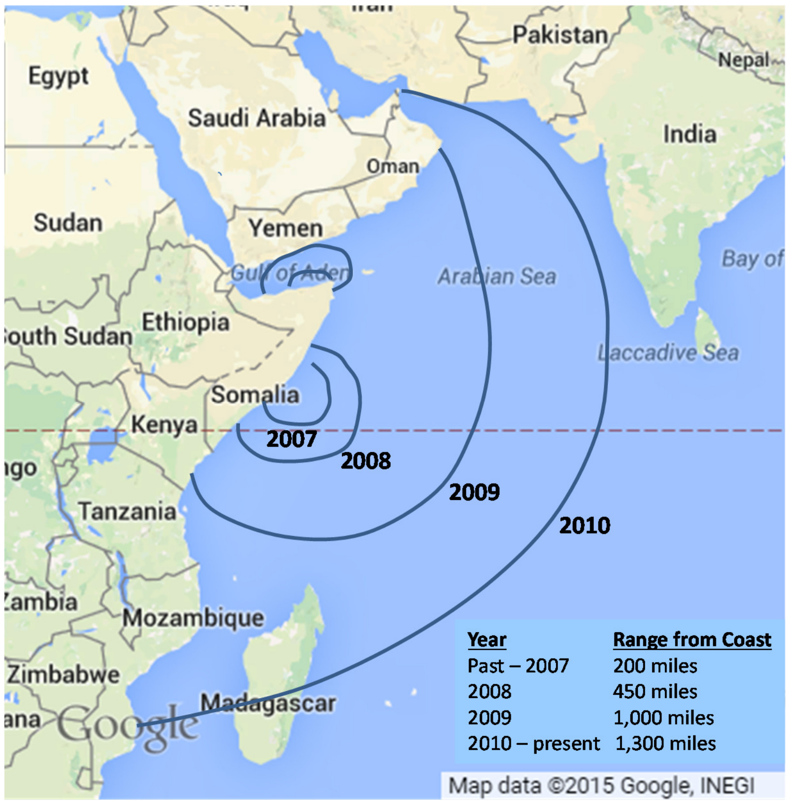
During 2013, due to extremely high levels of international policing efforts with military ships, especially in the Arabian Sea, the number of pirate attacks globally was down to its lowest level in years. Only 264 pirate attacks were reported globally in 2013, compared to an average of 377 for the five previous years (ICC-IMB 2014). However, the high level of military involvement in the region is not seen as either sustainable or cost-effective. The long range prognosis is that, once the military presence has diminished, Arabian Sea piracy will again flourish (Hunt 2012, Ng 2014).

1.2 The Spread of Somali Piracy

In many ways, modern-day Somalia presents the "perfect storm" for maritime piracy. Three main factors coincide to make Somalia attractive to pirates: ideal geographic location, permissive political environment, and extreme poverty (Chalk 2009).

Somalia is geographically located within close proximity to two of the world's busiest, most strategic maritime chokepoints: the Bab el-Mandeb and the Straits of Hormuz. These chokepoints control access to the Red Sea and the Persian Gulf, respectively. All ship traffic to or from these destinations must transit the Arabian Sea in relatively close proximity to Somali waters. With the advent of Mother-Ship and Pirate Action Group tactics, to be discussed later in section 1.3 of this paper, pirates are now able to range more than 1,000 miles from the Somali coast, rendering no locations in the Arabian Sea safe from potential attack (Bridger 2011). The geographical expansion of Somali pirate attacks has widened over the past decade as tactics have evolved (Figure 2).

Politically, Somalia remains among the most lawless, decentralized nations in the world today. Throughout most of Somalia, local warlords reign and invoke a "might is right," anything-goes style of government. In the south of Somalia, large swaths of the country are controlled by al-Shabaab, a cell of the militant Islamist terrorist group al-Qaeda (Murphy 2009). Not only do local leaders condone piracy, they actively encourage it, as they are typically paid a percentage of the revenues. For most of those in power in Somalia, piracy is looked at as a source of funding, which allows them to stay in power.



Year	Range from Coast
Past – 2007	200 miles
2008	450 miles
2009	1,000 miles
2010 – present	1,300 miles

Figure 2: Expansion of Somali Piracy Attacks
(Source: Map from Google Maps. Data adapted from Bridger 2011.)

Somalia is extremely poor. Average annual income is $140 (World Vision Inc. 2014). With extreme poverty comes a nothing-to-lose mentality among Somali youth. Though a high percentage of those who engage in piracy drown at sea (Archibugi and Chiarugi 2009), it is still seen as the most attractive option for financial well-being among many Somali youth. Just as the poor in America are the most frequent players of the lottery, the poorest youth in Somalia are the most prone to play the "piracy lottery," hoping to be among the few who beat the odds and strike it rich. Despite high mortality rates at sea and international efforts to try captured pirates in court, Somalia has a seemingly endless supply of young people willing to risk the odds.

1.3 Maritime Piracy Tactics

Maritime piracy requires a significant, land-based support infrastructure in order to succeed. Pirate land bases readily provide supplies, personnel, and intelligence. A land support network is critical to holding hostages and negotiating for their release. In Somalia, as previously discussed, organized crime, war lords, and potential terrorists (al Shabaab) are all willing and able sponsors of maritime piracy, as they look to it as a source of much-needed funds. Many authorities have suggested that the only way to permanently get rid of Somali piracy is to somehow get rid of pirate land bases within the country (Murphy 2009).

A modern-day "pirate ship" is typically a 20 to 30 foot fishing skiff equipped with oversized outboard engines (Figure 3). A typical skiff is manned by 4 to 6 young men, usually less than 25 years old, who are heavily armed with Rocket Propelled Grenades (RPG's) and automatic weapons. Two or three skiffs typically work together, navigating via GPS and communicating via hand-held radio and satellite phone, and potentially staying far offshore for days at a time. Pirates often seek first to attack a particular ship that their land-based intelligence network has identified as having a valuable cargo or being vulnerable to attack (Chalk 2008). Other times, or after a planned attack has not succeeded, pirates at sea seek out any target of opportunity.

When a potential target ship has been spotted, pirates race to it at top speed. A combination of stealth and speed are critical at this point. Approach is typically from the rear, taking advantage of the radar shadow behind the ship to remain unseen. Pirates often work together in Pirate Action Groups (PAG's) of two or three small boats. One or two boats will maneuver alongside or in front of the target vessel, often firing heavy

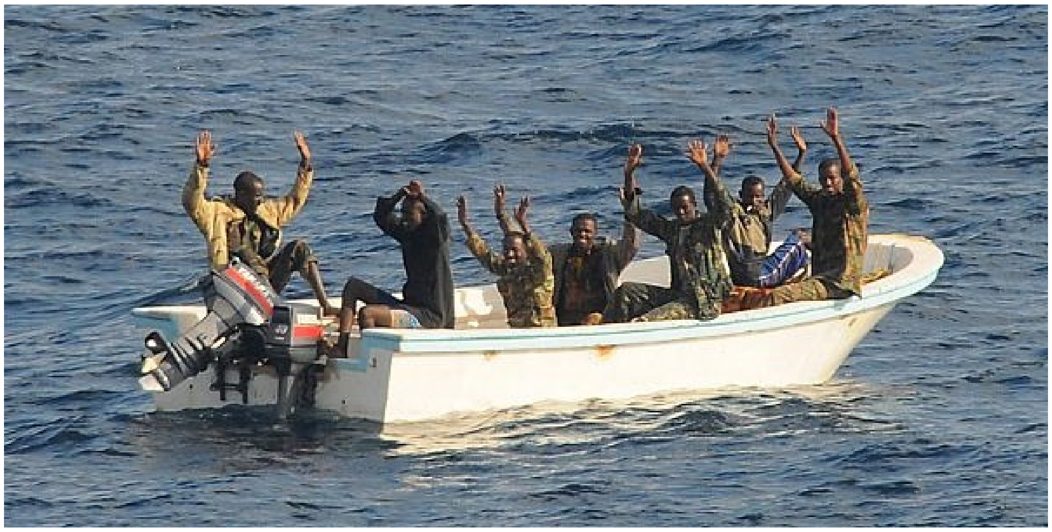
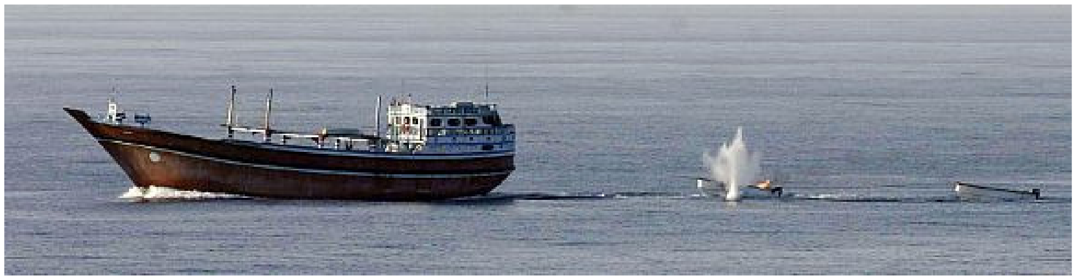


Figure 3: Typical Somali Pirate Skiff
(Source: U.S. Navy)

weapons at it, in an effort to slow the target and distract its crew. Meanwhile, the final small boat approaches the target vessel undetected from its stern, and pirates use ladders to rapidly and clandestinely board the ship. Once onboard, pirates quickly move to the ship's bridge, using whatever force is necessary to gain control of the ship (Whiteneck 2011). The pirates slow the ship, embark additional pirates, then round up and secure the crew. At this point, pirates have complete control of the ship, and they navigate it to the coastal waters of Somalia, where the ship and crew are held hostage until a ransom is negotiated and paid.

On average, Somali pirates successfully board and gain control of only about 20% of the ships they target (Bridger 2011). Section 1.4 discusses some of the countermeasures that contribute to this low success rate. Thus, about 80% of the time, after an unsuccessful or thwarted boarding, the pirates find themselves alone, hundreds of miles out to sea, in a small and poorly outfitted skiff, with nothing to show for their effort, and in extreme peril. The reaction of most pirates is to immediately attempt to find a new target. If another ship is visible on the horizon, they race towards it and attempt to board, despite no foreknowledge of its cargo or crew (Whiteneck 2011).

One tactic that has proven especially effective for Somali pirates is the use of mother ships to increase their range and on-station time. Captured vessels of all sizes, from fishing dhows to oil tankers, are used as floating support bases, from which smaller skiffs can be launched. Pirates can stay on station longer and in worse weather, with significantly less fatigue, by using mother ships as staging areas from which to launch their attacks. In addition, the mother ships have better communication facilities to receive intelligence from shore and better radar to spot target ships while still over the horizon. Once a potential target vessel has been detected over the horizon, pirates use the mother ship's radar to track it, launch two or three small pirate skiffs, and then use radio communications to vector the skiffs towards the target for intercept. Using mother ships, pirates can reach virtually all corners of the Arabian Sea and stay on station for weeks at a time (Warbrick 2008). The innovation of using mother ships was responsible for the dramatic increase in range of pirate attacks shown in Figure 2. Before mother ships were used, all Somali pirate attacks occurred within 200 miles of the coast. Today, using mother ships, no region of the Arabian Sea is off limits, and pirate attacks have occurred as far as 1,300 miles from the Somali coast (Bridger 2011). Additionally, pirate mother ships can be extremely hard to detect, since they often look like regular shipping traffic (Figure 4).



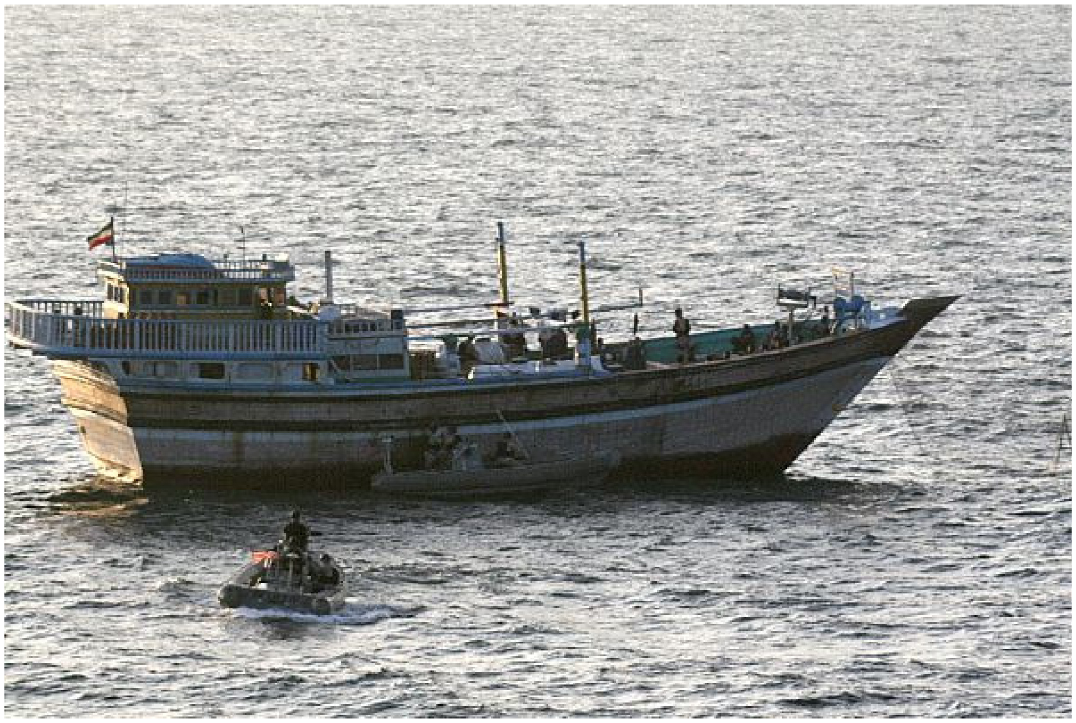


Figure 4: Pirate Mother Ships or Regular Traffic?
(Source: U.S. Navy)

1.4 Piracy Prevention: Countermeasures

Merchant ship captains and crew utilize an array of countermeasures in their attempts to thwart a pirate boarding. These methods range from fairly passive to highly aggressive and active.

Among the more passive of countermeasures are those dealing with a ship's course and speed. In general, if a ship maintains a speed of greater than 15 knots, its chances of being successfully boarded are dramatically reduced. In any significant sea state, pirate skiffs have a difficult time traveling at high speeds. Even when pirate skiffs are able to maintain higher speeds, their rate of closure on high speed contacts is so slow that they usually give up the chase before intercepting their target. Another passive method of pirate avoidance is evasive maneuvering, where the ship's course is frequently changed in an erratic pattern. A ship with frequent course changes is difficult for pirates to track and intercept, even when using a mother ship's radar facilities. During an attempted boarding, erratic course changes make boarding significantly more difficult and sometimes swamp and sink the pirate skiff (Hummel 2011).

Physical barriers and disruption devices are additional methods of thwarting unwelcome boarders. Chain-link fence and/or barbed wire are sometimes installed at locations of the ship with low freeboard, as these are the areas most vulnerable to rapid boarding. Disruption devices, such as fire hoses, slippery foam on the decks, and Long Range Acoustic Devices (LRAD), can further impede potential boarders (Figure 5). Though these barriers and devices might not hold off a determined boarding party indefinitely, they usually delay the boarding. This extra time can be used to maneuver the ship towards help or to secure the crew in a safe location onboard ship (Duda and Szubrycht 2009).

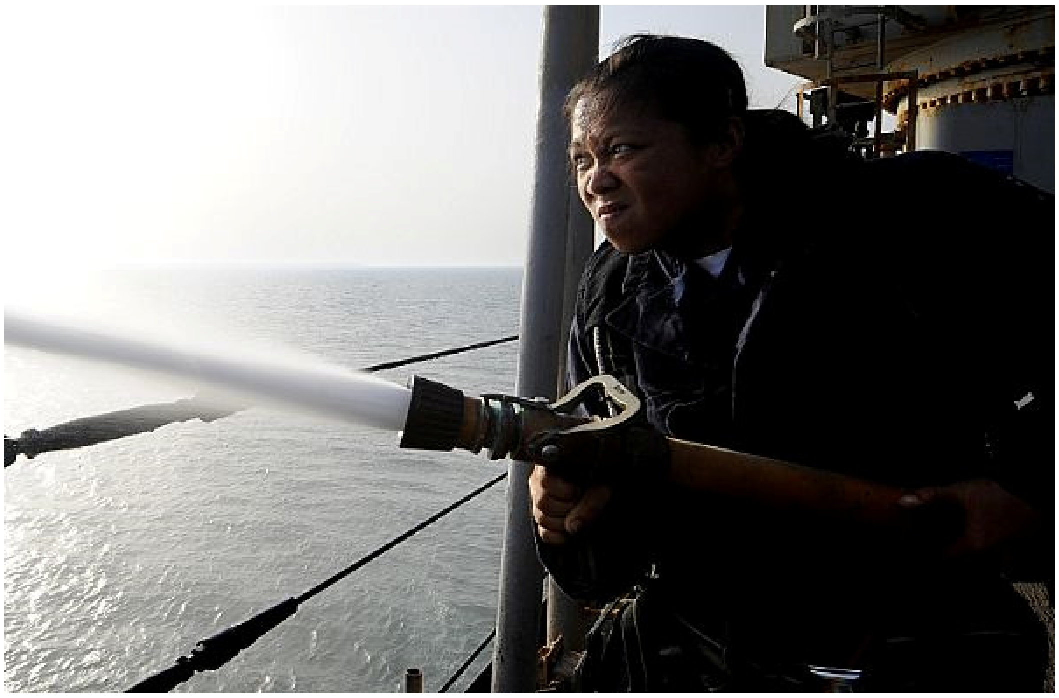


Figure 5: Disruption Devices May Slow or Thwart Potential Boarders
(Source: U.S. Navy)

Since 2010, most ship owners have begun to employ active measures to prevent Somali piracy. At a minimum, unarmed guards and extra lookouts are posted on virtually all ships transiting the Arabian Sea and West Indian Ocean. Many merchant ships now travel in convoys with other ships and via routes that will take them near military ships. Meanwhile, the military, especially the U.S. Navy, has successfully employed Unmanned Aerial Vehicles (UAV's) to locate, track down, and destroy pirate mother ships (La Boon 2011). Merchant ships are warned ahead of time to avoid areas where UAV's have spotted pirates recently, while naval warships speed to those areas to intercept the pirates.

A small but growing percentage of merchants have employed the most aggressive methods to thwart pirate attacks. Some ships have been outfitted with custom-built safehouses, called citadels, to which the crew can retreat if boarded by pirates. A locked citadel cannot be breached from the outside, and some citadels even have facilities within them to allow the crew to maintain steerage of the ship (Bateman 2012). Thus, even though pirates successfully board the ship, they are unable to gain control of the ship or its crew. The worst they can do is to steal items from it.

Finally, at the most aggressive end of the spectrum of countermeasures versus piracy, some ships have chosen to employ armed security guards and/or armed escorts for the transit through the most dangerous areas. Though costly, these methods have so far proven successful. To date, no vessels employing armed security guards or armed escorts have been successfully boarded. However, the relative sample size of potential target ships fitting this description is small. Also, concerns have been raised that, should pirates successfully board an armed merchant vessel, they will be more likely to retaliate with violence towards the crew (Mineau 2010).

Chapter 2: Piracy Prediction Methods

When asked about his strategy for batting, baseball Hall of Famer Willie Keeler replied, “Hit ‘em where they ain’t.” This philosophy also embodies the single best countermeasure versus piracy: avoid encountering them in the first place. The key to avoiding pirates is to know where they are (or are most likely to be) then travel in another direction. To this end, three main methodologies have emerged for predicting the likely locations of maritime pirates:

- Prediction based on past piracy problem areas
- Prediction based on geography and political factors
- Prediction based on agent-based simulation using intelligence and weather data, incorporating Bayesian inputs about recent pirate activity

Each of these methodologies is briefly discussed in the following pages.

2.1 High Threat Regions

The earliest and simplest method of piracy prediction is to treat an entire region as a high-threat or low-threat area, based on its past history of attacks. Within a given region, such as the combined Red Sea, Gulf of Aden and Arabian Sea area, no distinction is made regarding probability of pirate attack at individual locations. At any location within a high-risk geographic region, the likelihood of attack is treated as equally high. No distinction is made in the risk of being within a few hundred miles of the Somali coast, where most attacks historically have occurred, versus being over a thousand miles from the Somali coast, where only the most sophisticated of pirates can operate via use of mother ships. In other words, mariners transiting the region are essentially warned, “It’s all bad. Beware!”

For high-threat regions, anti-piracy planning charts are printed and distributed to mariners (Figure 6). These charts generally list anti-piracy advice applicable to the region and shade the entire region in which mariners should exercise caution (United Kingdom Hydrographic Office 2010). Because there is no distinction within a region as to level of threat at particular locations, and given that most crews transiting the waters near Somalia have already conducted anti-piracy training, the value of such charts is minimal.

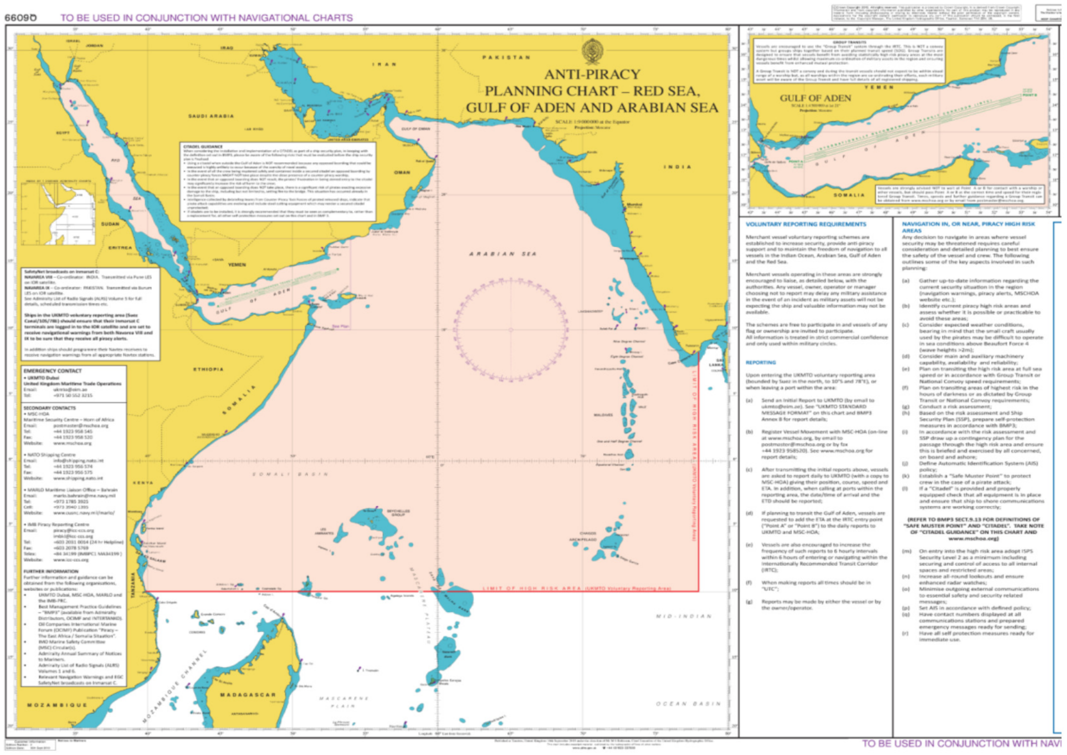


Figure 6: Anti-Piracy Planning Chart for the Arabian Sea Region
(Source: United Kingdom Hydrographic Office 2010)

2.2 Threat Density Plots

In 2010, researchers at Rutgers University conducted a study using historical information to predict where future pirate attacks would occur. This study, called Risk Terrain Modeling (RTM), was based on the premise that 3 factors could predict whether or not an area would experience pirate activity in the coming year: density of shipping routes, presence of maritime chokepoints, and the area's score in the global Failed States Index. From these 3 factors, they identified 7 layers of piracy risk, each of which was evaluated in a dichotomous (high risk / no risk) manner. Using these assessments, they created a piracy risk level for each area of the globe. Risk levels were color-coded then plotted on maps according to location.

Using risk data from 2008, the RTM model accurately predicted the global locations of maritime piracy incidents in 2009 (Moreto and Caplan 2010). By predicting future piracy hotspots, RTM could be a useful tool for policymakers. It might allow them to take political measures that prevent piracy's spread. However, RTM is not a tool that mariners can use on a daily basis. As with anti-piracy planning charts, the RTM model only tells mariners that an area is bad and should be avoided, not how to travel through it and avoid pirates. It gives no data on particular locations, nor does it give real-time predictions based on current information.

2.3 Forecasting via Simulation

In order to provide a more dynamic, real-time forecasting of pirate locations, both the United Kingdom and the United States developed prediction models that incorporate agent-based simulation. The three most successful models have been the United Kingdom Global Model (UKGM), the Piracy Performance Surface (PPS) model, and the Pirate Attack Risk Surface (PARS) model. Each of these three are discussed briefly in the following pages.

2.3.1 United Kingdom Global Model (UKGM)

In 2010, the United Kingdom developed a real-time forecasting model that utilizes meteorological data to predict the probable locations of pirate activity in the Arabian Sea. The United Kingdom Global Model (UKGM) is run four times per day and provides a rolling six day forecast. Inputs to the model include present and forecast weather, winds, sea states, and water temperatures. The UKGM is purely an agent-based simulation based on weather conditions. It does not incorporate real-time intelligence about known pirate activity and tactics. Even if pirates are reported at a particular location, the UKGM does not adjust its probability values (Ritchie 2010).

2.3.2 Piracy Performance Surface (PPS) Model

In 2009, the Naval Oceanographic Office developed the first real-time piracy prediction model, the Piracy Performance Surface (PPS) model. This model, which is used as a forward-warning tool by military and commercial vessels, graphically shows the predicted piracy threat level in the Arabian Sea region. The PPS model uses a combination of the environmental forecast and historical locations of observed pirate activity, as well as Bayesian input of recent pirate activity, to calculate a probability of real-time pirate activity in discrete areas of the Arabian Sea. Each area is a square box approximately 50 miles by 50 miles. When an attempted or successful pirate boarding is reported, the probability is raised for the next 48 hours at that location plus the bordering areas. These increased probability values slowly dissipate over the next 7 days (Slootmaker 2011).

2.3.3 Pirate Attack Risk Surface (PARS) Model

The success of the PPS model led to follow-on research to develop an even more accurate piracy forecasting tool, the Pirate Attack Risk Surface (PARS) model. The PARS model adds extra layers of input regarding real-time intelligence and pirate behavior patterns that the PPS model did not include (Slootmaker 2011). The output of PARS is a forecast of the probability of pirate presence as a function of latitude and longitude within the Arabian Sea region over time.

The piracy probability at each of over 2,000 locations in the PARS model is calculated based on 72 hour meteorological forecasts, real-time intelligence, and historically observed pirate behavior trends. Meteorological inputs include data on waves, winds, and tides. Intelligence inputs include information about recent pirate attacks, suspected mother ship spotting, and pirate base camp activity ashore. Historically observed behaviors include known areas of operation, current piracy tactics, and observed pirate behaviors after unsuccessful attacks.

The PARS model uses agent-based simulation along with these inputs to calculate a new probability surface every 12 hours, giving an updated probability for each location on the map. When real-time pirate attacks are reported, the PARS model uses this information as Bayesian input to further adjust its probability predictions for the coming week.

The PARS model was created using the Python 2.6.2 program; however, a MATLAB version of the model has also been developed. Currently, the PARS model is the only known piracy forecasting tool that combines meteorological and intelligence data, along with Bayesian input of recent activity. Its output is considered the most accurate predictor of pirate activity currently available (Slootmaker 2011).

To create its output piracy prediction map, the PARS model starts with a map of the Arabian Sea, such as that shown in Figure 7.

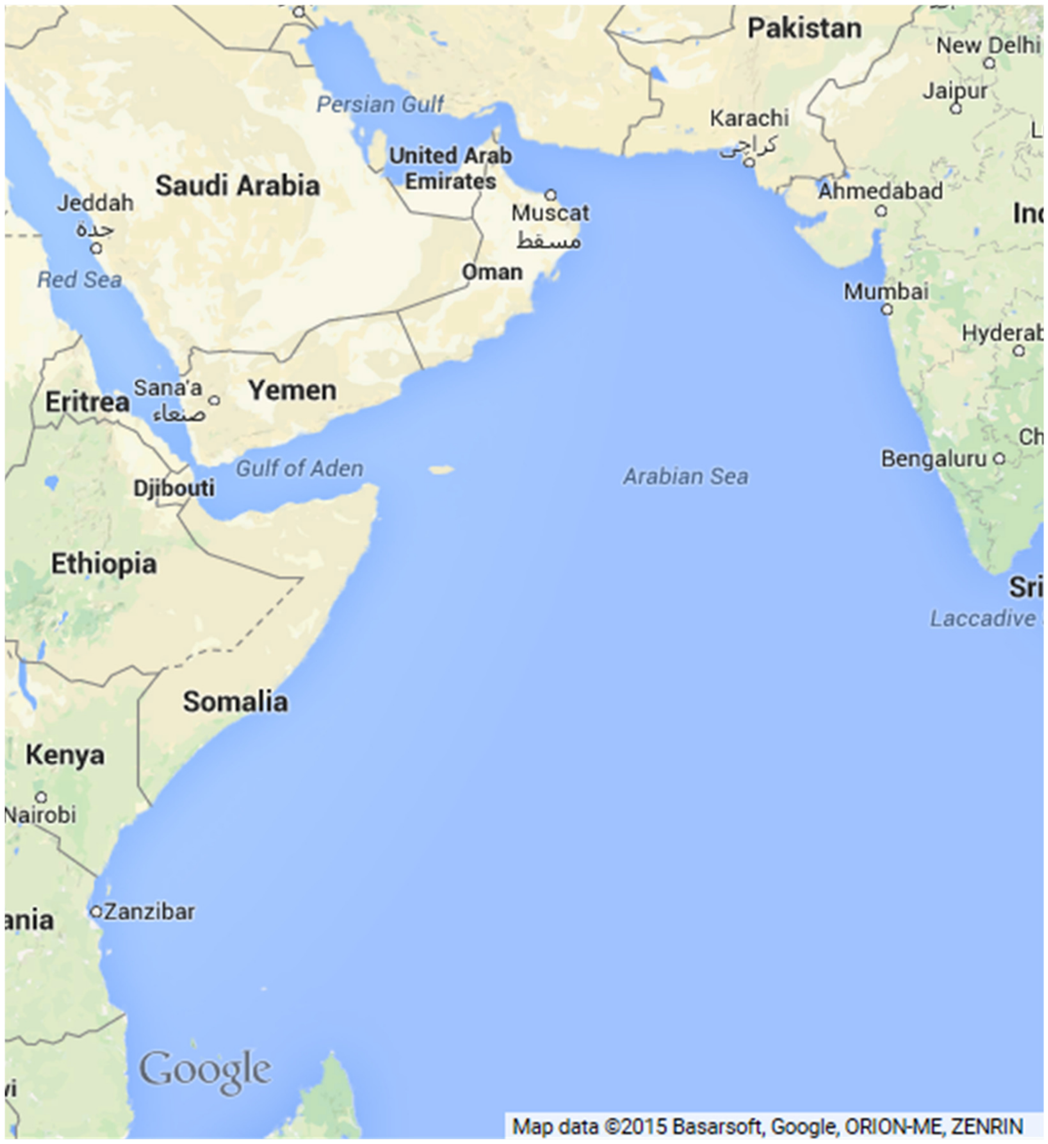


Figure 7: Map of Arabian Sea Region
(Source: Google Maps)

The PARS model then divides the Arabian Sea region into areas of 0.8 degrees latitude by 0.8 degrees longitude. This essentially creates a 43 x 50 matrix of 2,150 equally sized grid squares that are each 50 miles by 50 miles in size. The dividing of the region into grid squares is shown in Figure 8 .

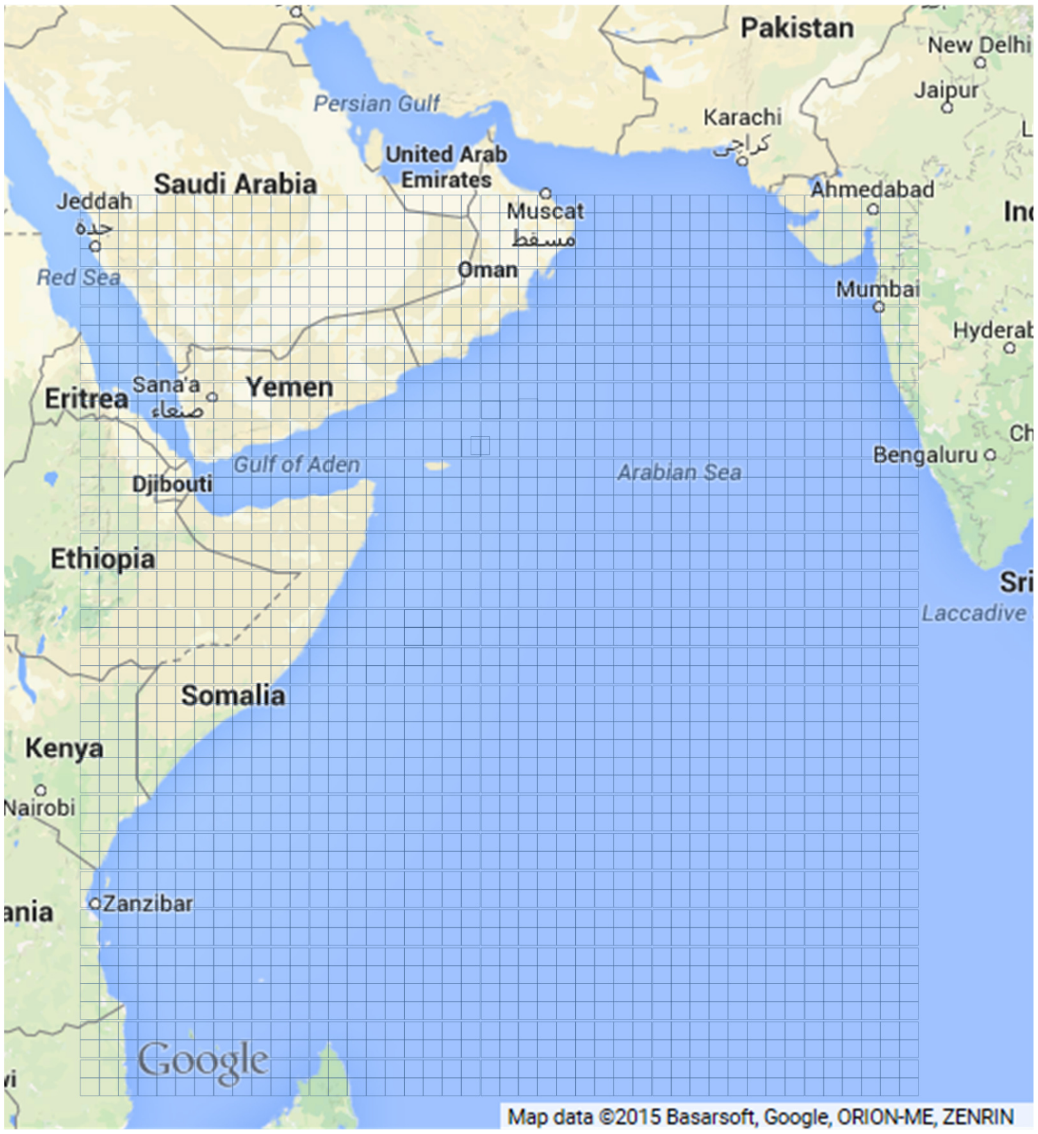


Figure 8: Dividing the Arabian Sea Region into 50 mile x 50 mile Areas
(Source: Map from Google Maps)

Finally, the PARS model calculates a unique probability value related to pirate activity in each grid square. As discussed above, the probability at each location is based on weather forecasts, intelligence reports, and pirate behavior trends. It is updated every 12 hours using Bayesian input of observed pirate activity. An example of the type of data generated in the PARS model is shown in Figure 9.

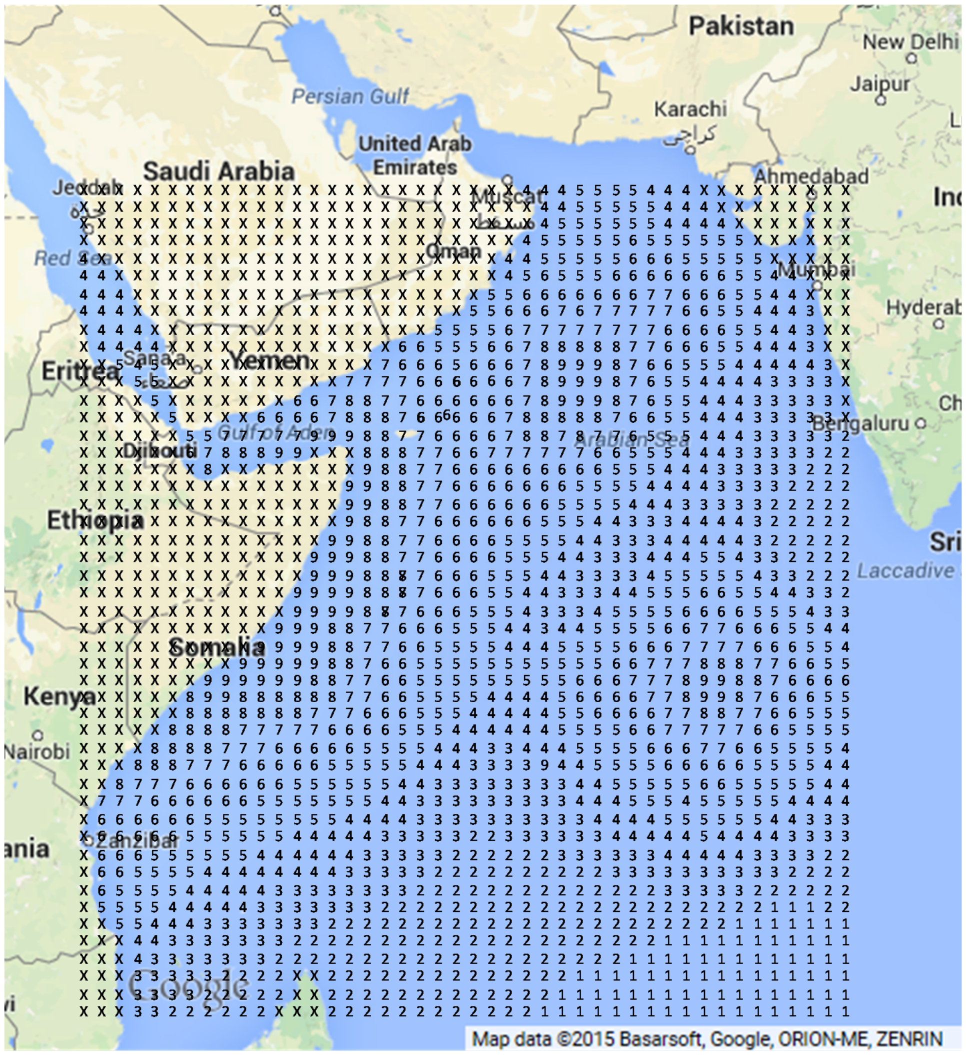


Figure 9: Example of Calculated Values at Each Location in PARS Model
(Source: Map from Google Maps. Data adapted from Slootmaker 2011.)

Typically, the graphical output of the PARS model does not include probability numbers. Rather, each square is color-coded according to its threat level. Red represents high pirate activity threat levels. The scale descends to orange, yellow, blue blue-green, and finally green for the lowest piracy threat levels. Figure 10 is an adapted example of the type of PARS output that the military or mariners might use to plan their transit routes.

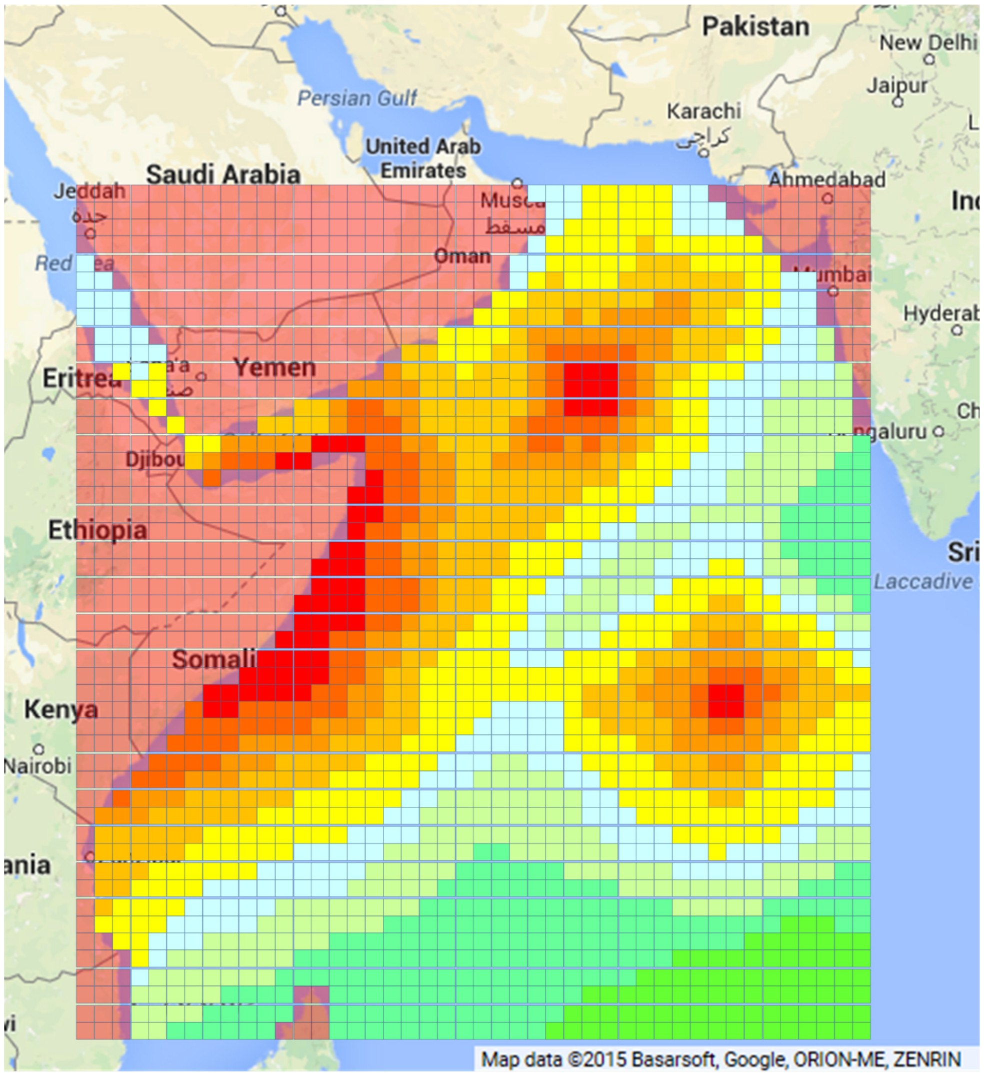


Figure 10: Adapted Example of PARS Model Output
(Source: Map from Google Maps. Data adapted from Slootmaker 2011.)

Chapter 3: Problem Definition

3.1 Pirate Avoidance: What's Still Missing?

The PARS model is the most accurate tool currently available for predicting the probability of encountering pirates in the Arabian Sea. It is updated twice daily to incorporate the most recent meteorological and intelligence data, and it provides a nearly real-time threat picture. The United States Navy and allied nations use the PARS model as a guide for directing their anti-piracy efforts. They focus their naval presence in the areas of highest predicted pirate concentration.

Unclassified outputs from the PARS model, color-coded into areas of lowest to highest threat levels, are sometimes made available to civilian mariners. In transiting the Arabian Sea region, using the PARS output, mariners can ensure that their course avoids the highest threat regions and that ample anti-piracy precautions are in place elsewhere.

However, mariners using the PARS model are more or less on their own to determine the "best" transit path. The tradeoff is that, by diverting to avoid high threat regions, they may spend more actual time in the "danger zone" and might spend significantly more on their operating and fuel costs. Is it worthwhile to go twice as far and spend twice as long in pirate-infested waters to avoid a high threat area? What about a medium threat area or a low-to-medium threat zone? Where should mariners draw the line on which regions to avoid and which to speed straight through at full speed? And how does transit cost enter into a mariner's decisions? Excessive diversions to avoid even the most miniscule threat of pirates will drive up costs and add delays. Should mariners on fast moving ships with anti-piracy countermeasures in place follow the same "best" route as those on slower moving ships with little to no countermeasures? What is missing from the PARS model is mapping of the optimum transit route through the Arabian Sea that minimizes the overall probability of encountering pirates, while at the same time attempting to reduce any extra costs associated with such diversions.

For example, suppose Figure 11 represents a matrix of probability values (in thousandths of one percent) while transiting a portion of the Arabian Sea. A ship, starting at point A with no foreknowledge of pirate locations, might follow the shortest distance route, represented by the solid black path across the middle of the matrix. They would travel a distance of 50 grid squares but risk a 1.538% chance of encountering pirates. Thus, their transit cost would be based on traveling 50 grid squares * 50 miles per grid square = 2,500 miles. However, there would also be an

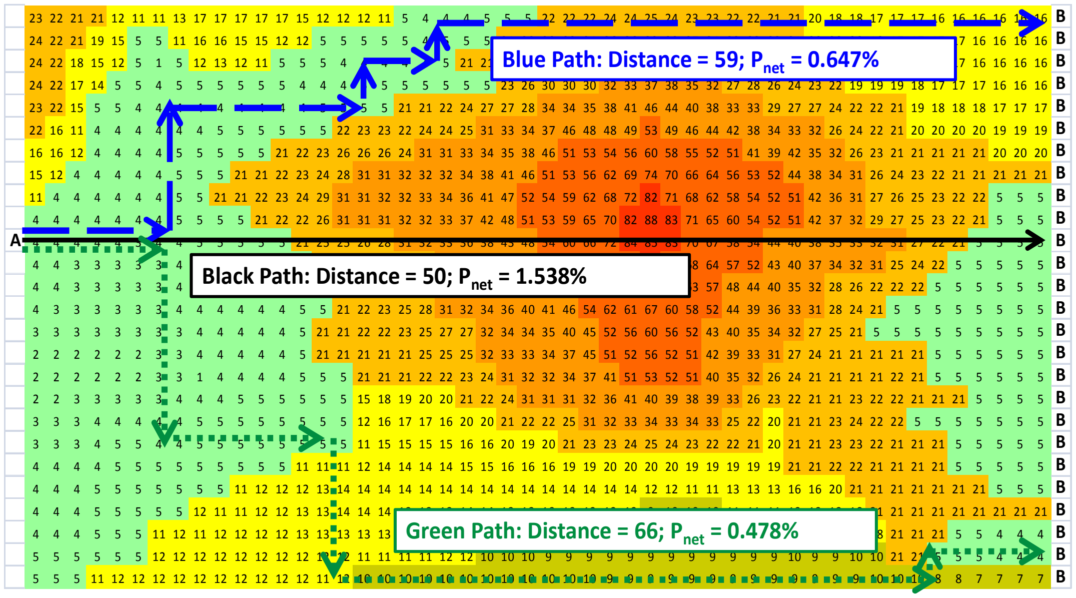
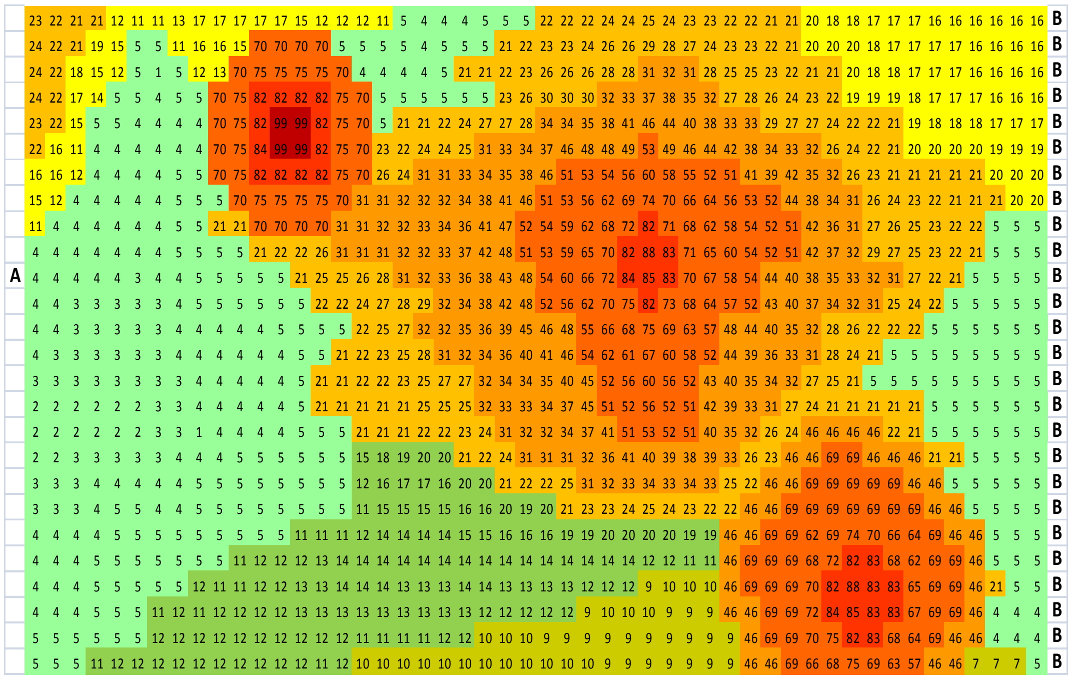

Blue Path: Distance = 59; P_{net} = 0.647%
Black Path: Distance = 50; P_{net} = 1.538%
Green Path: Distance = 66; P_{net} = 0.478%
A
B

Figure 11: Comparison of Routes Through the Piracy Prediction Matrix
(Values Shown are Probabilities in units of thousandths of one percent)

Expected Monetary Value (EMV) associated with their high piracy risk. Because a successful pirate attack would have negative financial impact in the millions of dollars, even a small piracy risk has a potential financial impact that must be considered.

Alternatively, another mariner, starting from the same location and with access to data about possible pirate locations, might choose to follow a route above or below the highest risk location. This would reduce his chances of encountering pirates even though it would increase his total distance traveled. In Figure 11, following the dashed blue path above the danger area increases the distance traveled to 59 grid squares = 2,950 miles while reducing the overall piracy risk to 0.647%. Traveling the even longer dotted green path increases the distance to 66 grid squares = 3,300 miles but further decreases the overall piracy probability to 0.478%.

For relatively simple threat pictures like the example just discussed, a mariner might be fairly confident that he is choosing close to the optimum course to lower his overall piracy risk, while still keeping transit costs down. However, for more complicated threat pictures with multiple active Piracy Action Group mother ships in the area, the choice of best transit path is not so obvious. The best path for pirate avoidance is not nearly as clear in Figure 12 as in the previous example. A tool that helps identify the optimum transit path to avoid pirate activity, while considering both transit cost and the Expected Monetary Value of the cost of a successful pirate attack, would be useful.



	23	22	21	21	12	11	11	13	17	17	17	17	17	15	12	12	12	11	5	4	4	4	5	5	5	22	22	22	24	24	25	24	23	23	22	22	21	21	20	18	18	17	17	17	16	16	16	16	16	16	B
	24	22	21	19	15	5	5	11	16	16	15	70	70	70	70	5	5	5	5	4	5	5	5	21	22	23	23	24	26	26	29	28	27	24	23	23	22	21	20	20	20	18	17	17	17	17	16	16	16	16	B
	24	22	18	15	12	5	1	5	12	13	70	75	75	75	75	70	4	4	4	4	5	21	21	22	23	26	26	26	28	28	31	32	31	28	25	25	23	22	21	21	20	18	18	17	17	17	16	16	16	16	B
	24	22	17	14	5	5	4	5	5	70	75	82	82	82	82	75	70	5	5	5	5	5	5	23	26	30	30	30	32	33	37	38	35	32	27	28	26	24	23	22	19	19	19	18	17	17	17	16	16	16	B
	23	22	15	5	5	4	4	4	4	70	75	82	99	99	82	75	70	5	21	21	22	24	27	27	28	34	34	35	38	41	46	44	40	38	33	33	29	27	27	24	22	22	21	19	18	18	18	17	17	17	B
	22	16	11	4	4	4	4	4	4	70	75	84	99	99	82	75	70	23	22	24	24	25	31	33	34	37	46	48	48	49	53	49	46	44	42	38	34	33	32	26	24	22	21	20	20	20	20	19	19	19	B
	16	16	12	4	4	4	4	5	5	70	75	82	82	82	82	75	70	26	24	31	31	33	34	35	38	46	51	53	54	56	60	58	55	52	51	41	39	42	35	32	26	23	21	21	21	21	21	20	20	20	B
	15	12	4	4	4	4	4	5	5	5	70	75	75	75	75	70	31	31	32	32	32	34	38	41	46	51	53	56	62	69	74	70	66	64	56	53	52	44	38	34	31	26	24	23	22	21	21	21	20	20	B
	11	4	4	4	4	4	4	5	5	21	21	70	70	70	70	31	31	32	32	33	34	36	41	47	52	54	59	62	68	72	82	71	68	62	58	54	52	51	42	36	31	27	26	25	23	22	22	5	5	5	B
	4	4	4	4	4	4	4	5	5	5	5	21	22	22	26	31	31	31	32	32	33	37	42	48	51	53	59	65	70	82	88	83	71	65	60	54	52	51	42	37	32	29	27	25	23	22	21	5	5	5	B
A	4	4	4	4	4	3	4	4	5	5	5	5	5	21	25	25	26	28	31	32	33	36	38	43	48	54	60	66	72	84	85	83	70	67	58	54	44	40	38	35	33	32	31	27	22	21	5	5	5	5	B
	4	4	3	3	3	3	3	4	5	5	5	5	5	5	22	22	24	27	28	29	32	34	38	42	48	52	56	62	70	75	82	73	68	64	57	52	43	40	37	34	32	31	25	24	22	5	5	5	5	5	B
	4	4	3	3	3	3	3	4	4	4	4	4	5	5	5	5	22	25	27	32	32	35	36	39	45	46	48	55	66	68	75	69	63	57	48	44	40	35	32	28	26	22	22	22	5	5	5	5	5	5	B
	4	3	3	3	3	3	3	4	4	4	4	4	4	5	5	21	22	23	25	28	31	32	34	36	40	41	46	54	62	61	67	60	58	52	44	39	36	33	31	28	24	21	5	5	5	5	5	5	5	5	B
	3	3	3	3	3	3	3	3	4	4	4	4	4	5	21	21	22	22	23	25	27	27	32	34	34	35	40	45	52	56	60	56	52	43	40	35	34	32	27	25	21	5	5	5	5	5	5	5	5	5	B
	2	2	2	2	2	2	3	3	4	4	4	4	4	5	21	21	21	21	21	21	25	25	25	32	33	33	34	37	45	51	52	56	52	51	42	39	33	31	27	24	21	21	21	21	21	5	5	5	5	5	B
	2	2	2	2	2	2	3	3	1	4	4	4	4	5	5	5	21	21	21	22	22	23	24	31	32	32	34	37	41	51	53	52	51	40	35	32	26	24	46	46	46	46	22	21	5	5	5	5	5	5	B
	2	2	3	3	3	3	3	4	4	4	5	5	5	5	5	5	15	18	19	20	20	21	22	24	31	31	31	32	36	41	40	39	38	39	33	26	23	46	46	69	69	46	46	46	21	21	5	5	5	5	B
	3	3	3	4	4	4	4	4	5	5	5	5	5	5	5	5	12	16	17	17	16	20	20	21	22	22	25	31	32	33	34	33	34	33	25	22	46	46	69	69	69	69	69	46	46	5	5	5	5	5	B
	3	3	3	4	5	5	4	4	5	5	5	5	5	5	5	5	11	15	15	15	15	16	16	20	19	20	21	23	23	24	25	24	23	22	22	46	46	69	69	69	69	69	69	69	46	46	5	5	5	5	B
	4	4	4	4	5	5	5	5	5	5	5	5	5	11	11	11	12	14	14	14	14	15	15	16	16	16	19	19	20	20	20	20	19	19	46	46	69	69	62	69	74	70	66	64	69	46	46	5	5	5	B
	4	4	4	5	5	5	5	5	5	5	11	12	12	12	13	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	12	12	11	11	46	69	69	69	68	72	82	83	68	62	69	69	46	5	5	5	B
	4	4	4	5	5	5	5	5	12	11	11	12	12	13	13	14	14	14	13	13	13	14	14	13	13	13	13	12	12	12	9	10	10	10	46	69	69	69	70	82	88	83	83	65	69	69	46	21	5	5	B
	4	4	4	5	5	5	11	12	11	12	12	12	12	13	13	13	13	13	13	13	13	12	12	12	12	12	9	10	10	10	9	9	9	46	46	69	69	72	84	85	83	83	67	69	69	46	4	4	4	4	B
	5	5	5	5	5	5	12	12	12	12	12	12	12	12	12	12	11	11	11	11	12	12	10	10	10	9	9	9	9	9	9	9	9	9	9	46	69	69	70	75	82	83	68	64	69	46	46	4	4	4	B
	5	5	5	11	12	12	12	12	12	12	12	12	12	12	11	12	10	10	10	10	10	10	10	10	10	10	10	10	9	9	9	9	9	9	9	46	46	69	66	68	75	69	63	57	46	46	7	7	7	5	B

Figure 12: Example of a More Complicated Piracy Prediction Matrix
(Values Shown are Probabilities in units of thousandths of one percent)

Furthermore, a merchant ship traveling at 20 knots will take about 4 days to transit the Arabian Sea. During this time, the PARS map will be updated 8 times, once every 12 hours, with the latest predictions. PARS maps do not usually change quickly – the effects of weather and intelligence are accretive, based on a rolling sample of recent information, not just one report. The only rapid changes to the PARS maps come from Bayesian input of recent pirate activity. The influence of these activity reports starts in a concentrated location then spreads and dissipates over the next 7 days. Because the PARS map can change during a ship's transit period, the tool for identifying the optimized path should also be able to adjust quickly for new probability matrices as well as current location of the ship.

Identification of the Optimized Transit Path must also consider the economic realities involved and allow a mariner to choose the priority he places on pirate avoidance versus cost minimization. Is it worth it to divert significantly to lower one's probability of encountering pirates if it means a longer distance traveled and, thus, a higher transit cost? Perhaps it is worth it if the threat of piracy drops significantly while the transit cost increases only marginally. But what about the opposite case – when the risk of piracy drops only slightly but transit costs go up significantly? The tool used for determining the Optimized Transit Path should allow for the mariner to make his own decisions about optimizing pirate avoidance versus minimizing overall cost.

Finally, when it comes to piracy, not all ships are created equal. Once pirates are encountered (the probability predicted by the PARS model), some ships have a much higher likelihood of being successfully boarded than others. A slow moving ship with low freeboard will be successfully boarded almost every time that pirates attempt to get on board. Meanwhile, a fast moving ship with high freeboard is extremely difficult for pirates to board. This difference between ship types and speeds should also be accounted for in determining the Optimized Transit Path through the Arabian Sea, especially when cost is an important factor.

3.2 Objective: Solving the Optimized Transit Path Problem

Ultimately, the goal of this research is to develop the framework of a "Pirate Avoidance System" that could be used by mariners to minimize the probability of encountering pirates, while at the same time following an economically viable transit path. In a fashion similar to a modern GPS advising drivers of the best path to avoid areas of heavy traffic, the Pirate Avoidance System would advise mariners of the best path to avoid pirates. Using probability data from the PARS model, the system would determine the Optimized Transit Path through the Arabian Sea with the lowest overall net probability of pirate activity, the lowest overall cost (incorporating transit cost plus Expected Monetary Value of potential piracy costs), or some weighted average of the two.

One of the challenges of solving the piracy avoidance problem is to determine the best algorithm and logic in order to calculate the minimum cumulative path of nodes through the piracy network. The piracy matrix problem is similar to other path applications in common use today. However, it does have some differences that may or may not make these other methods appropriate to its solution. An Optimized Transit Path algorithm, specifically aimed at solving the piracy matrix shortest path problem, should be developed and its performance compared to that which could be achieved with pre-existing algorithms.

In some ways, the piracy Optimized Transit Path problem resembles Critical Path Methods used in project scheduling. For the scheduling of a project, network scheduling techniques can be used to determine the earliest start, earliest finish, latest start, latest finish, total slack, and free slack for each activity. However, all of these calculations are based on the limitations imposed by the project's critical path, its path of maximum total duration (Gido and Clements 2012). In the case of the piracy Optimized Transit Path problem, the minimum total duration is desired.

Though the Optimized Transit Path problem is similar in many ways to network scheduling, there are a few important differences. Each area on the map can be treated like a node in a network diagram. However, instead of durations for each node, either the probability of piracy or an overall cost for each node is used. Also, instead of movement in the network being confined in direction by precedence logic, movement in the piracy Optimized Transit Path problem is much more complex. From any node in the problem, movement can be forwards, backwards, left, right, or diagonal. In a fairly simple matrix of nodes, such as a 4x4 or 5x5, each having its own cost per node, the minimum cumulative cost to move from one side to the other can be determined fairly simply by trial and error. However, as the matrix gets larger, the task of identifying the minimum path becomes exponentially more difficult. For extremely large matrices, like the 43x50 array of probability values produced by the PARS model, hand calculation of the optimum transit path is almost impossible. For example, just traveling forward through the matrix, with no backtracking allowed, there are $43 * 3^{49}$ possible paths through this network.

In other ways, this Optimized Transit Path problem resembles some of the Transportation and Logistics Problems, as well as Computer Routing Problems, studied in Operations Research and Computer Science, as will be discussed in Chapter 4. In many computer network designs, as well as transportation routing problems, the objective is to transit from one location to another via the shortest or quickest path possible. This class of problems is generally referred to as shortest path problems and is broken up into deterministic shortest path problems and stochastic shortest path problems. In deterministic shortest path problems, the distance from one node (location) to another is considered a known, fixed value. In stochastic shortest path problems, the distance from one node to another is treated as a probability distribution with a mean and standard deviation, which can either be constant or vary with time. Solutions to stochastic shortest path problems generally rely on a deterministic shortest path solution method to find the mean path value, then utilize stochastic methods to calculate variations in the mean (Cho 2003).

The Optimized Transit Path problem for maritime piracy can effectively be treated as a deterministic shortest path problem. In the unclassified version of the PARS output occasionally released to the public, the probability of encountering pirates at each node is represented as a fixed value for that 12 hour period, with no standard deviation reported. Likewise, its time variance is not repeatable or predictable, as it relies on the ever-changing weather, intelligence, and pirate activity picture of the moment. A brief survey of deterministic shortest path methods is included in Chapter 6 of this report.

One aspect of the piracy shortest path problem that is somewhat unique is that it is looking for the shortest path from any point on the "starting line" of one side of the matrix to any point on the "finish line" on the other side of the matrix. The fastest of traditional shortest path algorithms calculate the shortest path from one particular starting point to one particular finish point or to all other potential finish points. Because a mariner in the Arabian Sea will continue his transit after passing through the danger area, he typically does not have to start or finish at specific points in the matrix.

As long as he crosses the “finish line” to exit the area of piracy danger (the area represented by the PARS matrix), it is sufficient. Thus, to use potential point-to-point shortest path algorithms for the piracy problem, then algorithm would have to be run an extra (number of nodes on starting line) * (number of nodes on finish line) times to evaluate every potential shortest path through the matrix and determine the overall shortest. For the 43 x 50 PARS matrix, this could require running the algorithm an extra 43 * 43 = 1,849 times.

Other traditional shortest path algorithms calculate the shortest path between a specific node and every other node. Still others calculate the shortest path from every node in the matrix to every other node in the matrix. For the piracy problem, these two classes of algorithms might be overkill, making for more calculations than necessary.

What is needed is an algorithm that efficiently calculates the shortest path from any point on the “starting line” of the matrix to any point on the “finish line” at the other end. Using input from the PARS model, this algorithm would efficiently find the best transit path through the Arabian Sea to avoid pirates and minimize costs. The performance of this algorithm, specifically its required number of calculations and computing time, could then be compared to the performance of the most prevalent deterministic shortest path methods at solving the Optimized Transit Path problem for maritime piracy.

Once the necessary algorithm has been developed, the next objective of this research is to write a computer code that solves the Optimized Transit Path problem for avoiding pirates and minimizing overall cost, based on the most efficient algorithm for solving the problem. This code will utilize the methodology and logic discussed above, specifically applied to input data from the PARS model. It will solve for the optimized path – that with the lowest net probability of pirate activity, the lowest overall cost, or some weighted combination thereof - through the entire Arabian Sea region. As an output, the computer model should summarize the Optimized Transit Path via a listing of the recommended waypoints to follow.

In addition, the computer model should give quantitative feedback on the "cost" of deviating from the Optimized Transit Path, in terms of increased net probability and increased cost. Thus, for example, if a mariner chooses to follow a shorter path as far as physical distance, he will be aware of the increase in probability of encountering pirates associated with his choice or the increase in overall cost, when EMV of piracy costs is also considered.

Analysis of the computer code associated with solving the Optimized Transit Path problem should also explore the relationship between matrix size and both theoretical and actual computation time needed to solve the problem. For example, if the number of nodes in the matrix were doubled, would this change double the required solution time? Would it square the solution time? Or is there another relationship between matrix size and computing time? The worst case number of calculations required to solve the problem should be represented as a polynomial in terms of number of nodes and/or edges (connections between nodes) so that it can be compared to other potential solution algorithms. In addition, for a reasonably powered computer that might be found on a merchant ship, experiments should be run to quantify the expected solution time when a new PARS input is received.

The main deliverables that result from this research should be as follows.

1. Methodology that accounts for differences between ships, configurations, and defensive measures in predicting the success rate of pirates attempting to board and take control of the ship.
2. Methodology that allows optimization of both piracy avoidance and overall cost reduction. This methodology should allow a mariner to make choices about his priorities regarding these two, sometimes competing, objectives.
3. Algorithm that solves the piracy Optimized Transit Path problem to find the shortest path from any point on the "starting line" of the matrix to any point on the "finish line."
4. Computer code of the piracy Optimized Transit Path algorithm. This code will allow calculation of both theoretical and actual performance of the algorithm.
5. Comparison of the piracy Optimized Transit Path algorithm's performance to that of already existing shortest path algorithms when solving the piracy avoidance problem.

Chapter 4: Nodal Matrices in Other Shortest Path Applications

Nodal matrices are frequently used to model the workspace in a wide variety of fields. Just as with piracy prediction, in most of these applications, finding the shortest path through the matrix is an objective. Three common applications that utilize shortest path algorithms through nodal matrices are computer network routing, transportation and logistics optimization, and underwater search planning.

4.1 Computer Network Routing

Within computer science, each network of computers, routers, and their connections is modeled as a matrix. Every router is modeled as a node in the matrix. Each router node is connected by a link to every other router node that is directly attached to it, either by hard wire connection or wirelessly. Each of these router nodes, in turn, has links that directly connect it to those routers to which it is directly attached. Each link in the matrix has properties and metrics associated with it. Routers will use knowledge of these properties to help them determine the best route to a desired destination. Among the metrics used to determine the best path are length of transmission time, bandwidth, load, reliability, delay, and Maximum Transmission Unit (maximum size in bytes of a data unit that a link can process) associated with individual links, as well as number of hops, path cost, and communications cost associated with a potential path (Kurose and Ross 2004).

A typical computer network matrix will have a large number of nodes that are directly connected to only one, or just a few, other nodes. These represent individual computer workstations and routers, which are usually connected via a relatively slow,

low bandwidth connection. Many of these individual nodes will feed into an intermediate router, which is, in turn, connected to a main server via a significantly higher bandwidth connection than the stand-alone computers'. This main server will then connect out to other main servers via extremely high bandwidth, high speed connections. A simple diagram of part of a typical computer network matrix is illustrated in Figure 13.

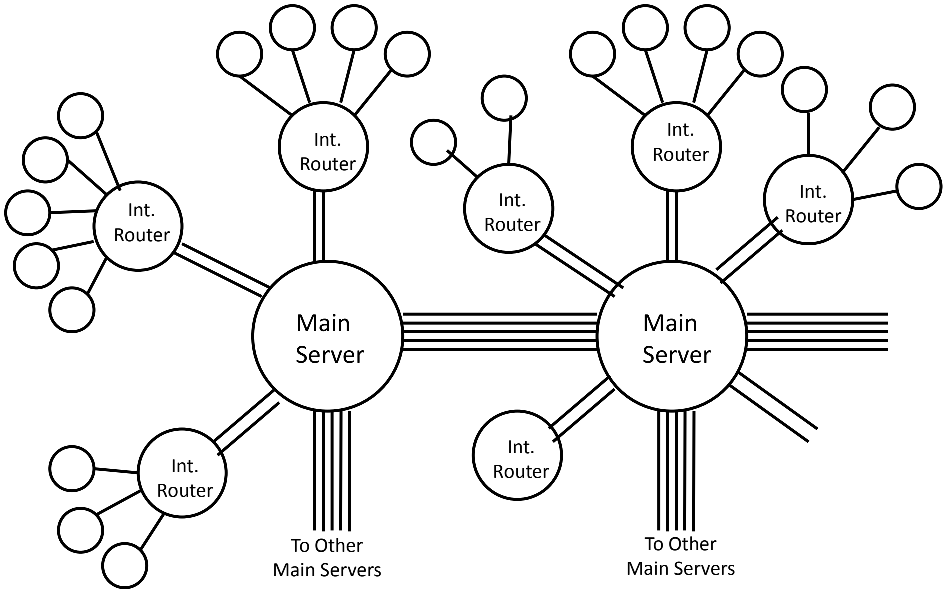


Figure 13: Simplified Diagram of a Typical Computer Network Matrix

In forming this vast array of routers and their connections into a matrix, two main protocols are followed for making updates to values in the matrix. In Distance Vector Routing Protocol, which is used as the basis for Routing Internet Protocol (RIP), an individual router informs only its immediate neighbors of any property changes in its links to other routers (Tanenbaum and Wetherall 2010). For example, if a connection between two routers were unplugged, only those routers directly connected to these two would be informed of the change. Because information about the link properties in locally held, computations related to link changes are much easier and less complex with the Distance Vector Routing Protocol than with other methods.

The Link State Protocol is one of these other methods. In it, a router sends a message to inform all other nodes in the matrix of any changes in link properties between it and a neighboring router. Thus, the Link State Protocol maintains information about link properties on a global level, with all nodes aware of the status of all other nodes in the network and each of their link connections (Cisco 2014). For large networks, this results in massive, complex computations any time a change is made anywhere in the network.

Once the network matrix is established, routing protocols are used to specify how one router will communicate with another. The routing protocol determines what path between tens, hundreds, or even thousands of router nodes that data will take in order to get from one location to a desired destination. Within the routing protocol, routing algorithms determine which specific route to choose in order to accomplish this data transfer.

With each routing protocol, the goal is essentially the same: to transfer data from one location to another location as quickly and efficiently as possible. To do this, a shortest path algorithm is used to determine the shortest length path between two nodes, based on whichever property of the route is being optimized. Usually, this optimization involves minimizing the net time to transfer data from one location to another. The overall length of a route is defined as the sum of the lengths of those links the data travels through while being transferred from one location to another (Kurose and Ross 2004). Two of the most common routing algorithms used in computer science network routing are Suurballe's Algorithm and the Edge Disjoint Shortest Path Algorithm.

Suurballe's Algorithm finds two different, distinct paths of minimum total length between a pair of nodes. Suurballe's Algorithm utilizes Dijkstra's Algorithm, which will be discussed in Chapter 6, to find the minimum length path in the matrix from one location to another. This path is recorded as the primary route. Suurballe's Algorithm then reweights each of the links on the shortest path, artificially increasing the lengths of all links that are part of the true shortest path. It then uses Dijkstra's Algorithm a second time to find a secondary or back-up shortest path through the matrix (Suurballe 1974 and Bhandari 1999). In routing data via Suurballe's Algorithm, the data is first sent via the primary shortest path. If, for some reason, the data does not get through to its destination, it is sent again, this time via the secondary path.

Edge Disjoint Shortest Pair Algorithm is another method sometimes used to solve for the shortest path through a computer routing network. Like Suurballe's Algorithm, it also calculates two shortest paths through the matrix. However, because the Edge Disjoint Shortest Pair Algorithm allows negative values for link lengths, it must use the significantly slower Johnson's Algorithm, which will be discussed briefly in Chapter 6, to calculate the shortest path (Bhandari 1999). Because its solution is slower and requires more computing power, the Edge Disjoint Shortest Pair Algorithm is not used as frequently as Suurballe's Algorithm in computer network routing applications.

4.2 Transportation and Logistics

The use of nodal matrices and shortest path calculations is also prevalent within transportation and logistics routing problems. GPS navigation software, which gives turn-by-turn directions to a car's driver, is perhaps the most commonly recognizable application in this field.

A car's navigation GPS uses a vector-based map to calculate the best route through a network of roads from one location to another, then translates this route into a set of directions for the driver to follow to his destination. For example, Figure 14 represents a nodal matrix used in GPS navigation. The lettered nodes in the figure represent specific locations on the map, and the links between them represent the roads that connect those locations. Expected driving time for each road is a metric associated with each link. If the quickest route from location A to location G were desired, the car's navigation system would consider all of the possible routes between these points

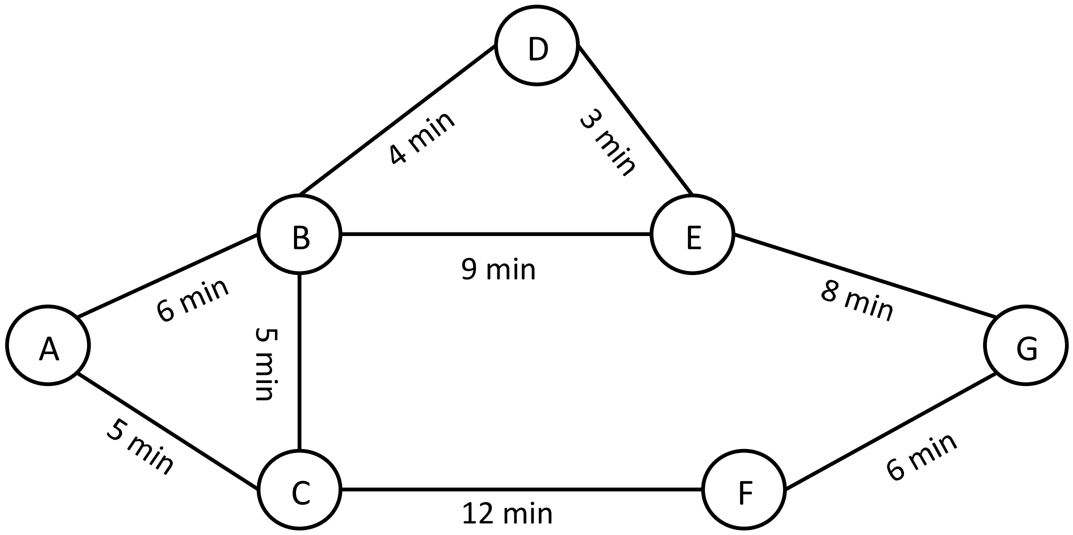


Figure 14: Example Transportation Network of Locations and Roads

then recommend the route with the shortest total time, route A-B-D-E-G, which has a route time of 21 minutes.

The key element of a vehicle's navigation system is the map database upon which calculations are run. A road map is essentially turned into a large matrix array. The basic elements of the map, such as addresses, locations, intersections, landmarks, and latitude/longitude coordinates, are treated as nodes of the matrix. Roads between nodes are treated as links and assigned properties, such as distance, typical travel time, and speed limit. Map providers put this matrix information of nodes and links into a standardized, well-defined, and documented format. Typical map matrix formats include Standard Interchange Format, Geographic Data Files (GDF), and proprietary forms of GDF (ISO 14825 2004).

Various parties, including the providers of GPS hardware, software, and services then handle this standardized map data. Many will add to it in proprietary ways, such as updating road travel times based on real-time traffic data. These parties typically transform the data into a proprietary run-time format before calculating a recommended path through the matrix, preventing interoperation of maps between different vendors' systems. Recently, a large number of vendors, including GPS manufacturers, data services, and automobile manufacturers, have begun an initiative aimed at standardizing the format of final map data used in car navigation systems, with the goal of increasing interoperability and sharing of data between systems (Flament 2005).

The methods by which different navigation system manufacturers handle their map data to recommend a route is proprietary to each. Differences include which shortest path algorithms are utilized, which characteristics of each link are prioritized to determine the recommended path, how backup routes are decided upon, and which roads are included in determination of the best route. However, all of the proprietary systems use some variant of traditional shortest path algorithms, most frequently Dijkstra's Algorithm, discussed in Chapter 6, as their backbone (Toth and Vigo 2001).

One common difference between navigation systems is the extent to which they pre-calculate major parts of the route. They will then use those pre-calculated sections rather than individual roads to decrease the size of the matrix and speed the time to calculate a recommended route between two locations. For example, the route for a cross-country journey from Washington D.C. to Los Angeles would primarily utilize predetermined paths between major waypoints, consisting mainly of interstates and large highways, rather than calculating every possible route using every possible road between the two locations. For example, in Figure 14, the navigation system might pre-calculate that the shortest route from B to E is via location D. Thus, depending on the requested destination, it might not even include the road directly from B to E in its consideration of overall shortest route.

4.3 Search and Recovery

Another field where nodal matrices are often used along with path calculations is in search theory, especially for underwater searches at sea. Search methods using probability mapping, Bayes Theorem, and path algorithms have been used successfully for a wide range of high-profile underwater searches, from lost ships and submarines to downed aircraft. In essence, Bayesian search methods direct the search first to areas where the probability of discovery is highest, then to regions of intermediate probability, and finally to areas of lowest probability. The search is continued until the probability of discovery is so low that the search is no longer economically viable (Stone 1975 and DeGroot 2004).

Bayesian search methods begin by formulating as many reasonable hypotheses as possible about the location of the missing object. For example, in the search for the sunken treasure ship *SS Central America*, different hypotheses about the ship's location were based on the captain's reported location at the time of sinking, the location of ships that picked up survivors from the sinking, and the results of simulations based on weather and currents on the day of the sinking (Stone 1992). For each hypothesis, a probability density function of likely locations of the missing object is created.

Meanwhile, for each location in the entire geographic region, the probability of detecting the missing object at each location, if it were actually located there, is determined. In an underwater search, the detection probability, when searching an object's true location, is a function of both the search method being used, the water depth at the location, and the size of the grid square represented by that location (PADI 2003). For example, divers and visual search methods will have a significantly higher likelihood of detecting a missing object in shallow water than they would in deeper locations. Both allowable bottom times and visibility drop quick as depth increases, greatly reducing the chances of spotting the object when visual methods are used. Other methods, such as towed array sonar search, have far less degradation of success probability as water deepens and are, therefore, typically a more preferred search method. The detection probabilities for each location are summarized on a second probability map of the entire geographic region, distinct from the probability map of likely locations of the missing object.

For each hypothesis, the probability density function of the object's location is then combined with the detection probability map, usually by multiplying the two values at each location (Stone 1975). For example, if the probability that a missing object is at a particular location is 2.0%, and the detection probability of the search method being used is 70% for that water depth and location grid size, the overall probability of actually finding the object at the location would be 2.0% * 70% = 1.4%. Performing a similar calculation for every location on the map creates an overall probability density map of the entire geographic region for each hypothesis. Essentially, this map shows the probability of discovering the object at each location on the map, if that location were searched, based on that hypothesis.

If there are missing hypotheses as to the missing object's location, the overall probability density maps for each hypothesis are then combined to form a single, overall, "best guess" probability map of the entire region. This is the probability map upon which the search pattern will be based. In combining probability density maps for the different hypotheses, a weighting factor is typically applied to each of the hypotheses (DeGroot 2004). Those hypotheses deemed more likely are given a higher weighting factor than those considered as more remote of possibilities. Thus, the more likely hypotheses make a greater contribution to the final, overall probability density map than the less likely ones.

Using the overall probability density map, a search path is chosen that will begin by searching locations of highest probability then slowly progress to locations of intermediate probability and finally to areas of lower probability (Stone 1975). The underwater scanning method to be used largely determines how this path is chosen and which, if any, path algorithms are used as part of the decision process.

For example, when towed array sonar is used in deep water locations, miles of cable, with the towed array at its end, might be spooled out behind the search vessel. With such a tow, the search vessel cannot make sharp turns for fear of entanglement. Thus, search paths through this matrix of probability values need to be relatively straight, with only minor course corrections along the way. Route selection algorithms used in this scenario need to encourage such a path.

On the other hand, a search vessel utilizing divers would have almost no restrictions on its search pattern. Because the vessel sits stationary for long periods of time while the divers are underwater then recovers them onboard before moving to the next search location, it has almost no restrictions on the pattern it follows from one search location to the next. For such a scenario, the search would likely progress from the highest probability location to the next highest to the next and so on.

Realistically, however, very few significant underwater searches today, especially the type that would create a probability density map, would utilize divers as an initial search method because it is far too slow of a process with too low of detection probabilities in deeper water. Almost all modern searches will incorporate some sort of towed array sonar, necessitating relatively straight paths through the nodal matrix of probability values.

In Bayesian search theory, once a location is searched and the missing object is not found there, the probability of discovery for that location, as well as all other locations in the matrix, is revised (Iida 1992). When the search of a location does not yield the missing object, the probability for that location is dramatically reduced, though usually not to zero. Correspondingly, the sum of probabilities for all other locations in the matrix is increased by the same amount. The process of revising probabilities is done according to Bayes Theorem.

Applying Bayes Theorem to the underwater search (Berger 1985), let A denote the event that the missing object is actually located at a given location. Let B denote the event that the object is not detected when that given location is searched. Suppose that the prior probability, P(A), that the missing object is at a particular location is p. This is the probability for that location from the overall probability density map, prior to the location being searched. The prior probability that the missing object is not at a particular location, therefore, is (1 – p). Also suppose that the probability of successfully detecting the object at a given location, if it is actually there, is d. Thus, the conditional probability, P(B|A), that the object will not be found at a location, even though the object is actually there, is (1 – d). Likewise, the initial prior probability of discovering the object at a given location is (p * d). Finally, the probability, P(B), that the missing object is not discovered at a given location equals the probability that it is not at that location, (1 – p), plus the probability that it is at that location but is not detected, p * (1 – d). Substituting each of the above values into Bayes Theorem allows a solution for the posterior conditional probability, P(A|B), of the probability that the object is at a given location, given that the location has been searched without finding the object. Let p’ designate the posterior probability for a given location.

Thus, for the following variables:

A =	event that object is located at a given location
B =	event that object is not detected at a given location when it is searched
p =	prior probability that object is at a given location
d =	probability that object will be detected at a given location if it is actually there
p’ =	posterior probability that object is at a given location, with the knowledge that the location has been unsuccessfully searched.

Bayes Theorem simplifies as follows:

$$P(A|B) = \frac{P(B|A) * P(A)}{P(B)} \qquad (1)$$

$$P(A|B) = \frac{P(B|A) * P(A)}{P(B)} \qquad (2)$$

$$p' = \frac{(1-d) * p}{(1-p) + p * (1-d)} \qquad (3)$$

$$p' = p\frac{(1-d)}{1-p*d} \tag{4}$$

Note that the revised, posterior probability, p', is always less than p.

Meanwhile, if the object was not discovered at a given location and the probability for that location was reduced, the probability of the object being at every other location is increased slightly. The sum of the increases in probability at all other locations combined equals the amount of decrease in probability at the location that was unsuccessfully searched. If the prior probability of the object being located at a given different location was q, then the revised probability for that location, q', would increase to

$$q' = q\ \frac{1}{1-p*d} \tag{5}$$

after the unsuccessful search of the first location. Thus, the revised probability for each other location, q', is greater than its prior probability.

During a search utilizing Bayesian search methods, the overall probability density map is constantly updated. With each unsuccessful search of a given location, the probability for that location is decreased, while the probabilities of all other locations are slightly increased. This process continues for as long as the search progresses, either until the object is found or until the search is called off.

Bayesian search methods were first developed in the 1966 search for a missing hydrogen bomb in the waters off of Palomera, Spain after a B-52 crash (Moody 2006). Since then, it has been used in a wide array of underwater searches, including those for the lost submarine USS Scorpion, a wide variety of treasure ships and historically significant wrecks, the flight recorder of the Air France Flight 447 disaster (Stone 2011). Bayesian search methods are also incorporated into the Computer Aided Search Program (CASP) used by the U.S. Coast Guard for at-sea search and rescue (Richardson and Discenza 1980).

Chapter 5: Incorporating Piracy Cost Considerations

5.1 Economic Considerations of Pirate Avoidance

The key feature that differentiates maritime piracy avoidance from other shortest path problems is the competition between two different objectives: minimizing the probability of encountering pirates and minimizing the total cost of transiting the Arabian Sea. Though sometimes in line, these two objectives are often in conflict with each other. When exiting the Persian Gulf, the shortest and most direct route south travels just off the coast of Somalia, where the chances of encountering pirates are typically highest. Thus, the transit route with the lowest operating costs is often the very same route with the highest probability of pirate attack. Mariners who want to lower their chances of encountering pirates often head east before turning south, putting hundreds of miles between them and the Somali coast. At the height of Somali piracy in 2011 and 2012, this made good business sense. The probability of piracy near the Somali coast was so great, and the associated costs of a successful attack, hijacking, and ransom were so high, that it made good business sense to travel extra distance in order to avoid the potential for a high-loss situation. However, today, when the rate of Somali piracy is but a shadow of its former self and the odds of being hijacked are significantly lower, such diversions might not make the best economic sense any longer.

On top of the changing piracy risk picture, some ships are far more prone to successful attack than others. If pirates are encountered, slow-moving ships with low freeboard height are far easier to board than faster ships with high freeboard. Thus, the best path to follow through the Arabian Sea, from an economic perspective, might not even be the same from one ship to the next. For a slower and lower ship, because the risk and cost of a successful boarding is so high, the best economic choice might be to take a significantly diverted path that avoids pirates as much as possible. Meanwhile, for a fast ship with tall sides, the best choice economically might be to travel a straight line path at top speed directly through the Arabian Sea, with the knowledge that their speed and height make them almost unboardable even if pirates are encountered.

Determining a "best" transit path is largely a matter of the priorities of the mariner. A path that lowers the chances of encountering pirates to its absolute minimum – good for the crew's morale – might yield one answer. A path that lowers the expected cost of the transit to its absolute minimum – good for the company's bottom line – might yield another answer. And a path that splits the difference between the two, putting some level of priority on pirate avoidance while still trying to keep transit costs low, might yield yet another answer.

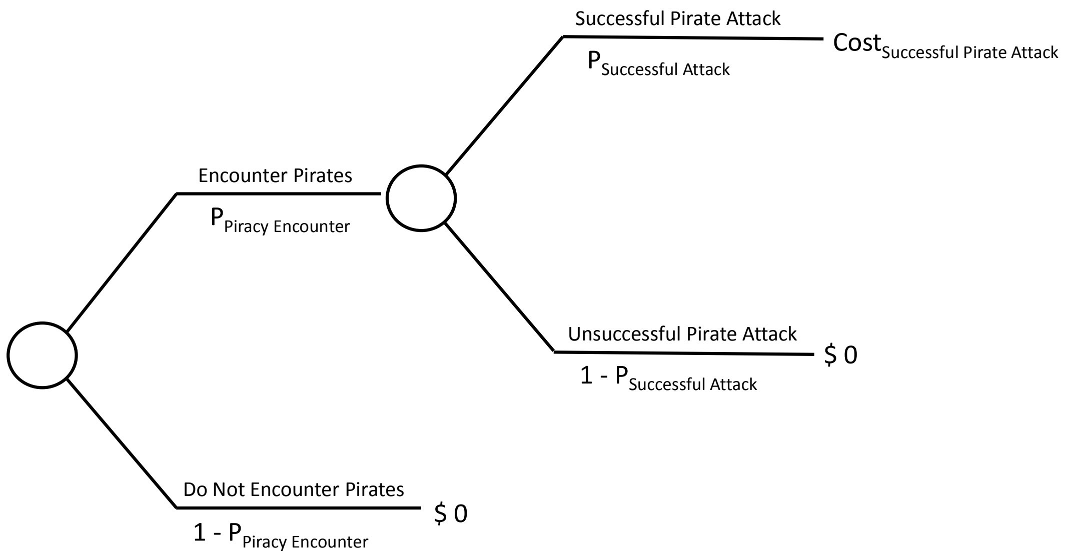
The PARS model gives a 43x50 array of piracy probability values within the Arabian Sea. The following sections discuss a methodology of transforming these probabilities into Expected Monetary Value (EMV) costs associated with successful pirate attacks as well as transit costs. By transforming the array into one of costs, weighting can be placed on those costs most important to the mariner, identifying the "best" transit path based on his priorities and goals.

5.2 EMV of Piracy Cost and Countermeasures in Each Node

The PARS model calculates the probability of encountering pirates in any of the 2,150 grid squares (43x50 matrix) that make up the Arabian Sea region. Each node value represents the probability of encountering pirates in that square, not the probability of being successfully attacked or the associated cost thereof.

Expected Monetary Value (EMV) is a valuable tool for determining the average cost, over time, when evaluating the financial impact of a series of related events, each of which has a probability of occurrence. From each uncertainty node in the EMV tree, designated by a circle, all of the potential events related to that event branch out. Each event has a probability. The sum of all probabilities of events emerging from an uncertainty node equals 1, or 100%. The EMV for each uncertainty node equals the sum for all events of the financial impact of each event times its probability of occurrence.

For the maritime piracy application, the EMV tree (Figure 15) has two uncertainty nodes. The first uncertainty is whether pirates will be encountered when transiting the Arabian Sea. Two events emerge from this node – pirates will be encountered or pirates will not be encountered. If pirates are not encountered, the financial impact on the mariner is zero. If pirates are encountered, the average financial impact is related to the EMV of the second uncertainty node – whether the pirates will successfully board the ship. If the pirates do not successfully board the ship, the financial impact is zero. However, if the pirates do board the ship, the financial impact is enormous, consisting of not only ransom costs but lost earnings from the ship while being held, as well as intangibles such as damage to the company's prestige or to the crew's morale. The


Successful Pirate Attack
$P_{\text{Successful Attack}}$
$\text{Cost}_{\text{Successful Pirate Attack}}$
Encounter Pirates
$P_{\text{Piracy Encounter}}$
Unsuccessful Pirate Attack
$1 - P_{\text{Successful Attack}}$
$ 0
Do Not Encounter Pirates
$1 - P_{\text{Piracy Encounter}}$
$ 0

$$\text{EMV}_{\text{Piracy}} = [\, \text{Cost}_{\text{Successful Attack}} * P_{\text{successful Attack}} + \$0 * (1 - P_{\text{successful Attack}}\ )\,] * P_{\text{Piracy Encounter}} + \$0 * (1 - P_{\text{Piracy Encounter}}\,)$$

$$\text{EMV}_{\text{Piracy}} = \text{Cost}_{\text{Successful Attack}} * P_{\text{successful Attack}} * P_{\text{Piracy Encounter}}$$

Figure 15: Expected Monetary Value Tree of Potential Piracy Outcomes

EMV related to pirates successfully boarding the ship equals the financial impact of a successful boarding multiplied by the probability that pirates will successfully board. For all transits, the overall EMV equals the probability of encountering pirates multiplied by this value.

To transform the PARS probability values into the Expected Monetary Value cost associated with piracy at each location ($\text{EMV}_{\text{piracy per node}}$), the probability of encountering pirates ($P_{\text{piracy encounter per node}}$) from the PARS model can be multiplied by the probability of a successful attack ($P_{\text{successful attack}}$)and the cost associated with a successful attack ($C_{\text{successful attack}}$).

$$\text{EMV}_{\text{piracy per node}} = P_{\text{piracy encounter per node}} * P_{\text{successful attack}} * C_{\text{successful attack}} \qquad (6)$$

Historically, only about 20% of the attacks attempted by pirates in the Arabian Sea have successfully resulted in the pirates boarding and capturing the ship (Bridger 2011). The rate of success is a combination of four main factors: the speed of the ship, the freeboard height of the ship, defensive measures in place on the ship, and the presence of armed guards and escorts. The probability of a successful attack can be estimated by determining a "knock-down" factor associated with the particular ship's configuration versus each of the above factors.

$$P_{\text{successful attack}} = K_{\text{speed}} * K_{\text{freeboard}} * K_{\text{defensive measures}} * K_{\text{guards}} \tag{7}$$

The tactics used by pirates to board a ship are similar, in many ways, to those used by U.S. Naval Special Warfare (SEAL) forces when interdicting noncompliant vessels on the high seas. The same factors that limit a pirate's ability to board a merchant vessel – speed, freeboard height, defensive measures, and armed guards – also limit a SEAL boarding party's ability to successfully board and stop a noncompliant ship. SEALs are significantly better trained and equipped and are able to overcome higher levels of adversity to their efforts. However, SEALs analyze these same four factors in determining a probability of successfully boarding a vessel and determining whether to proceed with such an operation.

Based on in-person interviews with numerous U.S. Navy SEALs, the impact of these four factors on a pirate crew's probability of successfully boarding a merchant vessel were estimated as summarized in Table 2 through Table 5. Note that the values in these tables are expert opinions, based on years of experience boarding non-compliant vessels, rather than being data-based values.

Table 2: Effect of Speed on Probability of Successful Pirate Attack
(Source: Interviews with Naval Special Warfare Personnel)

Vessel Speed (knots)	K_{speed}
< 10	1.0
10 to 15	0.9
15 to 20	0.6
20 to 25	0.3
> 25	0.1

Table 3: Effect of Freeboard Height on Probability of Successful Pirate Attack
(Source: Interviews with Naval Special Warfare Personnel)

Freeboard Height (ft)	$K_{freeboard}$
< 8	1.0
8 to 12	0.8
12 to 16	0.5
16 to 20	0.3
> 20	0.1

Table 4: Effect of Defensive Measures on Probability of Successful Pirate Attack
(Source: Interviews with Naval Special Warfare Personnel)

Defensive Measures	$\mathbf{K}_{\text{defensive measures}}$
None	1.0
Water Cannons	0.9
Deck Foaming	0.9
LRAD Sound System	0.9
Citadel without steering	0.7
Citadel with steering	0.4

Table 5: Effect of Guards and Escorts on Probability of Successful Pirate Attack
(Source: Interviews with Naval Special Warfare Personnel)

Guards and Escorts	$\mathbf{K}_{\text{guards}}$
None	1.0
Extra unarmed lookouts	0.9
Convoy with other vessels	0.8
Armed guards onboard	0.4
Armed escort vessel	0.1

Thus, if pirates are encountered, the probability of their successfully boarding a merchant ship can vary dramatically from one ship to the next. For example, an older, smaller cargo vessel might travel at 12 knots, have a freeboard height of 10 feet at the stern, possess no extra defensive measures, and employ extra unarmed lookouts when transiting the Arabian Sea. If such a vessel were attacked by pirates, as calculated with Equation 7, the estimated probability of a successful boarding would be quite high.

$$P_{\text{successful attack}} = K_{\text{speed}} * K_{\text{freeboard}} * K_{\text{defensive measures}} * K_{\text{guards}} \tag{7}$$

$$P_{\text{successful attack}} = 0.9 * 0.8 * 1.0 * 0.9$$

$$P_{\text{successful attack}} = 0.65$$

Meanwhile, a newer tanker will typically transit the Arabian Sea at just over 20 knots, have a minimum freeboard height of 18 feet, employ water cannons or deck foaming, and will frequently convoy with other similar vessels. Even if pirates were encountered, the chances of such a vessel being successfully boarded are very low.

$$P_{\text{successful attack}} = K_{\text{speed}} * K_{\text{freeboard}} * K_{\text{defensive measures}} * K_{\text{guards}} \tag{7}$$

$$P_{\text{successful attack}} = 0.3 * 0.3 * 0.9 * 0.8$$

$$P_{\text{successful attack}} = 0.07$$

The final component needed to determine an Expected Monetary Value of piracy attack in each node of the PARS matrix, $EMV_{\text{piracy per node}}$, is the cost associated with a successful pirate attack. As with the probability of successful boarding, the cost of a successful pirate attack can vary widely from one ship to another, depending on the type and size of the ship, the company that owns the ship, the amount of crew members involved, and the length of detainment of the ship, not to mention the outright brazenness and desperation of the pirate crew that captures it.

The direct monetary cost of an attack is a sum of multiple factors, most notably the cost of any ransom paid to the hijackers and the operating costs associated with the ship during the time of its capture. In 2011, average ransom costs were $4.9 million per ship, and hijacked ships were held an average of 2 weeks (Gardner 2012). Average operating costs for a non-U.S. flagged merchant vessel in the Arabian Sea, not including fuel costs, was $7,500 per day in 2011 (U.S. DOT Maritime Administration 2011). On average, a 14 day detainment will cost an additional 14 days * $7,500/day ≈ $100,000 of operating costs. Thus, a successful pirate attack will typically incur about $5 million of direct costs. Additional costs, somewhat harder to quantify, will include increased insurance premiums after an attack, lost opportunity cost of cargo the ship could otherwise be transporting during its detainment time, delayed schedules, decreased crew morale, and loss to company reputation and prestige.

Based on information about a particular ship's defenses against piracy and its associated cost of being successfully attacked, the PARS matrix of piracy probability values can be transformed into a matrix of Expected Monetary Value cost associated with piracy at each location.

As an example of calculating $\text{EMV}_{\text{piracy per node}}$, suppose the slow-moving, older cargo ship discussed earlier is traveling through a 50 mile by 50 mile region of the Arabian Sea for which the PARS model predicts that there is a 0.2% chance of encountering pirates. The Expected Monetary Value of piracy costs for this ship while transiting this region would be calculated as follows.

$$\text{EMV}_{\text{piracy per node}} = P_{\text{piracy encounter per node}} * P_{\text{successful attack}} * C_{\text{successful attack}} \quad (6)$$

$$\text{EMV}_{\text{piracy per node}} = 0.002 * 0.65 * \$5{,}000{,}000$$

$$\text{EMV}_{\text{piracy per node}} = \$6{,}500$$

The overwhelming chance is that there would be no piracy cost at all associated with this node, as pirates either would not be encountered or they would be unsuccessful in boarding. On extremely rare occasions, however, a successful pirate attack would occur and cost $5 million. The average piracy cost associated with this ship transiting this particular node, $\text{EMV}_{\text{piracy per node}}$, would be $6,500.

Thus, each node in the matrix will have a projected cost of piracy associated with it. If avoiding costs of piracy is a mariner's only priority, he should follow a path through the Arabian Sea that minimizes the net EMV of piracy associated with it. If piracy avoidance is the only consideration, this path will be identical to finding the path through the PARS matrix with the lowest net sum of probability values.

5.3 Transit Cost to Travel Through Each Node

The cost to transit a vessel through each 50 mile by 50 mile node in the PARS matrix is the sum of its daily operating costs per node plus its fuel costs per node.

$$C_{\text{transit per node}} = C_{\text{daily ops per node}} + C_{\text{fuel per node}} \tag{8}$$

Daily operating costs for a ship at sea includes the costs of its crew salaries, stores and lubes, maintenance and repair, insurance, and overhead. On average, these costs add up to $7,500 for foreign-flagged vessels and $20,000 for U.S.-flagged vessels (U.S. DOT Martitime Administration 2011). Multiplication by the size of each node in the PARS matrix (50 miles), division by the ship's speed (miles/hr), and simple units conversion can translate this value into the cost of daily operations associated with transiting each node.

$$C_{\text{daily ops per node}} = C_{\text{daily ops per day}} * \text{Node Size} * 1\text{ day}/24\text{ hr} * 1/\text{Speed} \tag{9}$$

$$C_{\text{daily ops per node}} = (\$/\text{day}) * (50\text{ miles/node}) * (1\text{ day}/24\text{ hr}) * (\text{hr/miles})$$

$$C_{\text{daily ops per node}} = \$ / \text{node}$$

Fuel costs per node can likewise be calculated my multiplying the fuel consumption (Metric Tons/Day) by the fuel cost ($/Metric Ton) and Node Size (50 miles), division by the ship's speed (miles/hr), and simple units conversion.

$C_{\text{fuel per node}}$ = Fuel Consumption*$C_{\text{fuel per day}}$*Node Size*1 day/24 hr*1/Speed (10)

$C_{\text{fuel per node}}$ = (MT/day) * (\$/MT) * (50 miles/node) * (1 day/24 hr) * (hr/miles)

$C_{\text{fuel per node}}$ = \$ / node

As an example of calculating transit cost per node, a typical 8,000 container ship traveling at 21 knots burns 125 metric tons of fuel in one day in order to transit 500 nautical miles (White 2010). In January 2015, the average cost of one metric ton of fuel was approximately \$350/metric ton (TSA Carriers 2015). Using the above data along with a daily operating cost of \$7,500, the cost to transit each node is calculated as follows.

$C_{\text{transit per node}} = C_{\text{daily ops per node}} + C_{\text{fuel per node}}$ (8, 9, 10)

$C_{\text{transit per node @ 21 knots}}$ = (\$7,500/day)*(50 miles/node) * (1 day/24 hr) * (hr/21 miles)

+ (125 MT/day) * (\$350/MT) * (50 miles/node) * (1 day/24 hr) * (hr/21 miles)

$C_{\text{transit per node @ 21 knots}}$ = \$744/node + \$4,340/node

$C_{\text{transit per node @ 21 knots}}$ = \$5,084/node

Note that shipping speed has a huge impact on the transit cost per node. For the same 8,000 container ship analyzed above, the fuel consumption drops to 58 Metric Tons/ Day if the speed is dropped to 15 knots and all other variables are held the same (White 2010). This reduction in speed from 21 to 15 knots reduces the transit cost per node by 24%, from \$5,084/node to \$3,862/node as follows.

$$C_{\text{transit per node}} = C_{\text{daily ops per node}} + C_{\text{fuel per node}} \tag{8, 9, 10}$$

$$C_{\text{transit per node @ 15 knots}} = (\$7{,}500/\text{day})*(50\ \text{miles/node}) * (1\ \text{day}/24\ \text{hr}) * (\text{hr}/15\ \text{miles})$$
$$+ (58\ \text{MT/day}) * (\$350/\text{MT}) * (50\ \text{miles/node}) * (1\ \text{day}/24\ \text{hr}) * (\text{hr}/15\ \text{miles})$$

$$C_{\text{transit per node @ 21 knots}} = \$1{,}042/\text{node} + \$2{,}820/\text{node}$$

$$C_{\text{transit per node @ 21 knots}} = \$3{,}862/\text{node}$$

In addition, the cost of fuel is highly variable over time and can likewise have a huge impact on transit cost per node. For example, fuel costs are currently about \$350/metric ton. However, in 2012, fuel costs averaged over \$700/metric ton (TSA Carriers 2015). Thus, just three years ago, fuel cost per node was double its current cost at all speeds.

5.4 Weighted EMV of Pirate Avoidance vs. Transit Cost

A mariner transiting the Arabian Sea faces competing demands. Does he try to minimize his chance of encountering pirates? Or does he try to minimize the overall costs of the transit? Or does he seek some middle ground between the two? Based on their motivations regarding piracy and transit cost, mariners fall into four categories.

1. If a mariner's only goal was to absolutely minimize his transit costs, he would travel in a straight-line path of the shortest possible route through the Arabian Sea, with no consideration at all for the possibility of pirate attack.

   $$\text{Chosen Route}_{\text{Category 1}} = \text{Route with } \min(\textstyle\sum C_{\text{transit}}) \qquad (11)$$

   Such a mariner would spend the smallest amount possible on operating and fuel costs. However, he would run the risk of incurring a significant cost if his ship were successfully attacked and boarded by pirates.

2. Meanwhile, if a mariner's only goal was to avoid pirates at all cost, he would choose to follow a path that follows those nodes with the smallest total sum of piracy probabilities in the PARS matrix. This would be the same path as if the mariner followed those nodes with the lowest total sum of $EMV_{\text{piracy per node}}$.

   $$\text{Chosen Route}_{\text{Category 2}} = \text{Route with } \min(\textstyle\sum EMV_{\text{piracy per node}}) \qquad (12)$$

Such a mariner might follow a route that diverts hundreds of miles from the shortest distance route through the Arabian Sea, in the interests of reducing his chances of a pirate encounter.

3. Other mariners, who are aware of the potential costs of piracy but still wary of increasing their transit costs too much, might choose to follow a route that minimizes the total overall cost of their route, where $EMV_{\text{actual total for route}}$ equals the sum of $EMV_{\text{piracy per node}}$ plus the sum of transit cost per node for all the nodes on that route. For each node on the route, its total EMV is as follows.

$$EMV_{\text{actual total per node}} = EMV_{\text{piracy per node}} + C_{\text{transit per node}} \tag{13}$$

Meanwhile, the EMV of an overall route is the sum of piracy cost and transit cost of nodes on that route.

$$EMV_{\text{actual total for route}} = \sum EMV_{\text{piracy per node}} + \sum C_{\text{transit per node}} \tag{14}$$

A mariner who wishes to minimize his total costs would choose the route that minimizes $EMV_{\text{actual total for route}}$.

$$\text{Chosen Route}_{\text{Category 3}} = \text{Route with } \min(EMV_{\text{actual totalfor route}}) \tag{15}$$

4. Finally, still other mariners might wish to strike a balance between trying to avoid pirates and minimizing their overall costs. Such mariners might be willing to divert a little bit, even at an increased total cost, if it means a significant reduction in their likelihood of pirate attack. However, they are not willing to make massive course changes, at significant extra cost, if it only has a minimal impact on reducing the overall probability of a pirate encounter. Such mariners recognize that there are qualitative factors associated with a successful pirate boarding, such as impact on crew morale and company prestige, not captured in EMV_{piracy} but that does have a tangible value. And, thus, they place a slightly higher value on minimizing the chances of pirate attack than they do on minimizing overall cost.

$$\text{Chosen Route}_{\text{Category 4}} = \text{Route with min(Weighted sum of } [EMV_{piracy} + C_{transit}]) \quad (16)$$

In order to reconcile between the competing motivations of mariners described above, a weighting factor, X, can be implemented into the calculation of EMV for each node as follows.

$$EMV_{\text{weighted total per node}} = EMV_{\text{piracy per node}} + X * C_{\text{transit per node}} \quad (17)$$

In transiting the Arabian Sea, mariners would follow a route that minimizes the sum of $EMV_{\text{weighted total per node}}$ from the nodes that are traveled through.

$$EMV_{\text{weighted total for route}} = \sum EMV_{\text{weighted total per node}} = \sum EMV_{\text{piracy}} + \sum (X * C_{\text{transit}}) \qquad (18)$$

Weighting Factor, X, is a number between 0 and 1 that represents the priority a mariner places on minimizing the chance of a pirate boarding versus minimizing the overall cost to travel across a region. The value of X is a subjective choice, based on the mariner's preferences regarding pirate avoidance.

$X = 0$ represents a mariner who wishes to avoid pirates at all costs, such as Category 2 above. In the case of $X = 0$, $EMV_{\text{weighted total per node}} = EMV_{\text{piracy per node}}$ for all nodes in the matrix. In following a route that minimizes $EMV_{\text{weighted total for route}}$ with $X = 0$, a mariner would be following the route that minimizes his chances of encountering pirates.

$X = 1$ represents a mariner who wishes to minimize his total overall cost, with consideration of the fact that there is an EMV cost associated with piracy in each node, such as Category 3 above. In the case of $X = 1$, $EMV_{\text{weighted total per node}} = EMV_{\text{piracy per node}} + C_{\text{transit per node}}$ for all nodes in the matrix. Thus, in following a transit route that minimizes $EMV_{\text{weighted total for route}}$ with $X = 1$, a mariner would be following the route that minimizes $EMV_{\text{actual total for route}}$, which is the optimized decision based purely on immediately tangible financial considerations.

$0 < X < 1$ represents a mariner who places an increased importance on piracy avoidance versus strictly short-term financial considerations, but not at the total exclusion of these considerations, such as Category 4 above. The closer that X is to 0, the higher the priority that the mariner places on piracy avoidance. Meanwhile, the closer that X is to 1, the higher the priority that the mariner places on minimizing total cost. Based on a mariner's choice of weighting factor, X, the optimized route will be that with the minimum value of $\mathrm{EMV}_{\text{weighted total for route}}$.

$$\text{Optimized Route}_{\text{Category 4}} = \text{Route with } \min(\mathrm{EMV}_{\text{weighted total for route}}) \tag{19}$$

$$\text{Optimized Route}_{\text{Category 4}} = \text{Route with } \min(\sum \mathrm{EMV}_{\text{piracy}} + \sum (X * C_{\text{transit}})) \tag{20}$$

The actual cost of transiting this optimized route, however, is still based on Equation 14 for the nodes on that route.

$$\mathrm{EMV}_{\text{actual total for route}} = \sum \mathrm{EMV}_{\text{piracy per node}} + \sum C_{\text{transit per node}} \tag{14}$$

Thus, based on a mariner's choice of X, his optimized transit path will be somewhere between the routes of minimum piracy probability and minimum overall cost. Note that the weighted $\mathrm{EMV}_{\text{weighted total for route}}$, which is used to choose the route, will always be less than or equal to the actual monetary cost of transiting that route, $\mathrm{EMV}_{\text{actual total for route}}$.

5.5 Example of Applying Weighted EMV to Transit Route Decisions

The following example demonstrates how the recommendation for Optimized Transit Path will change based on the priority that a mariner places on piracy avoidance versus overall transit cost.

Suppose that a 5 x 5 matrix of piracy probability values from the PARS model is as shown in Figure 16.

$$\begin{tikzcd}
2.7 & 2.4 & 1.2 & 0.3 \arrow[r] & 0.3 \\
2.7 & 1.8 & 0.9 & 0.3 \arrow[u] & 0.9 \\
2.7 & 0.3 \arrow[r] & 0.3 \arrow[r] & 0.3 \arrow[u] & 2.1 \\
0.9 & 0.3 \arrow[u] & 0.9 & 0.9 & 3.0 \\
0.3 \arrow[r] & 0.3 \arrow[u] & 1.2 & 2.1 & 3.0
\end{tikzcd}$$

Note: Piracy Probability Values are in One-Thousandths

Figure 16: Example 5x5 Array of Piracy Probability Values

Each of the values in matrix represents the probability of encountering pirates in thousandths. Thus, a value of 2.7 in the top-left node of the matrix represents a probability of 0.0027 that pirates will be encountered within that 50 mile by 50 mile region of the Arabian Sea during a transit. By inspection, one can determine that the route represented in the figure by arrows would have the minimum sum of node values for any route from the left side of the matrix to the right. This is based on the assumption that all movement through the matrix must be either in an up-down or a left-right direction, so that all moves represent the same physical number of miles transited, 50 miles. For this example, diagonal movement is not considered, since it would involve distances per node of greater than 50 miles.

Using the methods discussed in Section 5.2, the piracy probability values of the PARS matrix could be transformed into Expected Monetary Value costs of piracy associated with each node in the matrix. The first example ship discussed in Section 5.2 was an older, smaller cargo vessel travelling at 12 knots, with a freeboard height of 10 feet at the stern, possessing no extra defensive measures, and employing extra unarmed lookouts when transiting the Arabian Sea. For such a vessel, the methods of Section 5.2 would transform the 5 x 5 matrix of piracy probability values to the matrix of $EMV_{\text{piracy per node}}$ values in Figure 17.

Each of the values in the matrix represents the Expected Monetary Value cost due to piracy, measured in thousands of dollars. By inspection, the Optimized Transit Path route to minimize the cost of piracy, without any consideration of transit cost, is shown by arrows in the matrix. This is an identical route to that through the matrix of piracy probability values.

$$\begin{tikzcd}
9 & 8 & 4 & 1 \arrow[r] & 1 \\
9 & 6 & 3 & 1 \arrow[u] & 3 \\
9 & 1 \arrow[r] & 1 \arrow[r] & 1 \arrow[u] & 7 \\
3 & 1 \arrow[u] & 3 & 3 & 10 \\
1 \arrow[r] & 1 \arrow[u] & 4 & 7 & 10
\end{tikzcd}$$

Note: $\text{EMV}_{\text{Piracy per Node}}$ Values are Thousands of Dollars

Figure 17: Example 5x5 Array of $\text{EMV}_{\text{Piracy per Node}}$ Values

However, in addition to potential piracy costs, a mariner transiting the region represented by this matrix would also incur transit costs, consisting of both operating and fuel costs. As discussed previously in Section 5.3, for the vessel considered in this example, typical transit costs would be about $4,000 per node. A 5 x 5 matrix of transit costs per node for this ship would be as shown in Figure 18.

Each of the values in the matrix represent the transit cost to travel the full 50 miles distance across the node, in thousands of dollars. Because all of the node values are the same, the Optimized Transit Path route to minimize transit costs, with no consideration of piracy costs, would be a straight line across the matrix of the shortest possible distance. Any of the routes shown by arrows in the matrix would be equally good, as all are of the same minimal length.

$$\begin{tikzcd}
4 \arrow[r, dotted] & 4 \arrow[r, dotted] & 4 \arrow[r, dotted] & 4 \arrow[r, dotted] & 4 \\
4 \arrow[r, dotted] & 4 \arrow[r, dotted] & 4 \arrow[r, dotted] & 4 \arrow[r, dotted] & 4 \\
4 \arrow[r, dotted] & 4 \arrow[r, dotted] & 4 \arrow[r, dotted] & 4 \arrow[r, dotted] & 4 \\
4 \arrow[r, dotted] & 4 \arrow[r, dotted] & 4 \arrow[r, dotted] & 4 \arrow[r, dotted] & 4 \\
4 \arrow[r, dotted] & 4 \arrow[r, dotted] & 4 \arrow[r, dotted] & 4 \arrow[r, dotted] & 4
\end{tikzcd}$$

Note: $C_{\text{transit per Node}}$ Values are Thousands of Dollars

Figure 18: Example 5x5 Array of $C_{\text{transit per Node}}$ Values

The actual Expected Monetary Value total cost associated with each node in the 5 x 5 matrix would be the sum of the EMV_{piracy} values in Figure 17 plus the $C_{transit}$ values in Figure 18. These values are summed in Figure 19 below to produce a 5 x 5 matrix of $EMV_{actual\ total\ per\ node}$ values.

$$\begin{tikzcd}
13 & 12 & 8 & 5 & 5 \\
13 & 10 & 7 & 5 & 7 \\
13 & 5 \arrow[r, dashed] & 5 \arrow[r, dashed] & 5 \arrow[r, dashed] & 11 \\
7 \arrow[r, dashed] & 5 \arrow[u, dashed] & 7 & 7 & 14 \\
5 & 5 & 8 & 11 & 14
\end{tikzcd}$$

Note: $EMV_{Actual\ Total\ per\ Node}$ Values are Thousands of Dollars

Figure 19: Example 5x5 Array of $EMV_{Actual\ Total\ per\ Node}$ Values

Each value in the matrix represents thousands of dollars of actual EMV cost that could be expected in passing through that node. Note that the Optimized Transit Path route for minimizing actual total cost, as shown by arrows in the matrix, is of a shorter distance than the recommended route for piracy avoidance but longer than the route for minimizing transit cost. In this recommended path, it is still economically viable to make some course changes to avoid pirates. However, because transportation costs are now considered, only those course changes with the most significant economic impact are included in the recommended route.

If a weighting factor, X, is applied to the calculation of EMV for each node, it will produce a recommended route somewhere between the optimized routes for avoiding pirates and minimizing overall cost. Not surprisingly, a choice of X = 0.5 in the example 5 x 5 matrix produces an Optimized Transit Path recommendation that "splits the difference" between the piracy avoidance (X = 0) route and the minimized cost (X = 1) route, as demonstrated in Figure 20. Note that the values in this matrix are weighted values, $\text{EMV}_{\text{weighted total per node for X = 0.5}}$, and do not represent actual costs. To

$$\begin{tikzcd}
11 & 10 & 6 & 3 & 3 \\
11 & 8 & 5 & 3 \arrow[r, dashed] & 5 \\
11 & 3 \arrow[r, dashed] & 3 \arrow[r, dashed] & 3 \arrow[u, dashed] & 9 \\
5 \arrow[r, dashed] & 3 \arrow[u, dashed] & 5 & 5 & 12 \\
3 & 3 & 6 & 9 & 12
\end{tikzcd}$$

Note: $\text{EMV}_{\text{Weighted Total per Node – X = 0.5}}$ Values are Thousands of Dollars

Figure 20: Example 5x5 Array of $\text{EMV}_{\text{Weighted Total per Node for X = 0.5}}$ Values

calculate the actual cost of traveling the recommended route, one would have to sum the $EMV_{\text{actual cost per node}}$ values for each of the recommended nodes on the route.

Meanwhile, a mariner who put higher priority on piracy avoidance but wasn't ready to fully commit to whatever course changes were necessary, might choose a weighting factor of $X = 0.25$. In the example 5 x 5 matrix, this produces a recommended route that is closer to the piracy avoidance route (Figure 21). Note that, in this example, both the lower left corner and upper right corner of the matrix have alternative route recommendations, where either route would produce the same sum of node values on the route, $EMV_{\text{weighted total for route for } X = 0.25}$. Thus, there are four potential shortest routes through the matrix, each of which has the same sum of node values.

$$\begin{tikzcd}
10 & 9 & 5 & 2 \arrow[r, dashed] & 2 \\
10 & 7 & 4 & 2 \arrow[r, dashed] \arrow[u, dashed] & 4 \\
10 & 2 \arrow[r, dashed] & 2 \arrow[r, dashed] & 2 \arrow[u, dashed] & 8 \\
4 \arrow[r, dashed] & 2 \arrow[u, dashed] & 4 & 4 & 11 \\
2 \arrow[r, dashed] & 2 \arrow[u, dashed] & 5 & 8 & 11
\end{tikzcd}$$

Note: $\text{EMV}_{\text{Weighted Total per Node} - X = 0.25}$ Values are Thousands of Dollars

Figure 21: Example 5x5 Array of $\text{EMV}_{\text{Weighted Total per Node for } X = 0.25}$ Values

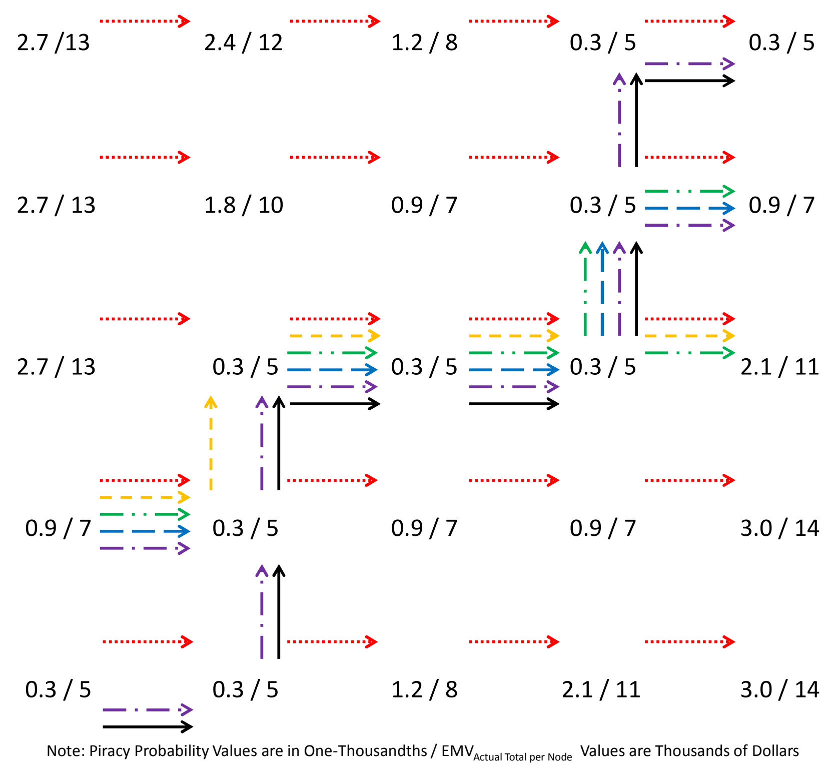
Likewise, a mariner whose highest priority was on minimizing overall cost but still wanted to put a little extra priority on pirate avoidance, might choose a weighting factor of X = 0.75. In the example 5 x 5 matrix, this produces a recommended route, based on $\text{EMV}_{\text{weighted total for route for } X = 0.75}$, that is closer to the piracy avoidance route (Figure 22). As with the previous example, the far right side of the matrix produces two alternative route recommendations, where either route would produce the same sum of node values.

$$\begin{tikzcd}
12 & 11 & 7 & 4 & 4 \\
12 & 9 & 6 & 4 \arrow[r, dashed] & 6 \\
12 & 4 \arrow[r, dashed] & 4 \arrow[r, dashed] & 4 \arrow[r, dashed] \arrow[u, dashed] & 10 \\
6 \arrow[r, dashed] & 4 \arrow[u, dashed] & 6 & 6 & 13 \\
4 & 4 & 7 & 10 & 13
\end{tikzcd}$$

Note: $\text{EMV}_{\text{Weighted Total per Node – X = 0.75}}$ Values are Thousands of Dollars

Figure 22: Example 5x5 Array of $\text{EMV}_{\text{Weighted Total per Node for X = 0.75}}$ Values

Depending on a mariner's priorities regarding piracy avoidance versus cost minimization, the Optimized Transit Path recommendation might be different from one vessel to the next. However, the tools developed in this section give mariners an objective means of quantifying the tradeoffs in their decisions. The six potential choices for optimized routes through the example 5 x 5 matrix that were analyzed are summarized in Figure 23.



LEGEND

⇢ = Route to Minimize Transit Cost

⇢ = Route to Minimize Actual Total Cost (X = 1)

⇢ = Route Weighted Towards Minimized Actual Cost (X = 0.75)

⇢ = Route Weighted Evenly Between Piracy and Cost (X = 0.5)

⇢ = Route Weighted Towards Pirate Avoidance (X = 0.25)

→ = Route to Minimize Piracy Cost (X = 0)

Figure 23: Optimized Routes Vary Based on Choice of Priorities

A summary of the net piracy probability and the financial implications of each of these potential routes is summarized in Table 6.

Table 6: Summary of Optimized Route Choices

Route Description	Symbol	$\sum P_{piracy}$	$\sum EMV_{piracy}$	$\sum C_{transit}$	$EMV_{actual\ total\ for\ route}$
Minimize Transit Cost – Row 1	············→	6.9	23	20	43
Minimize Transit Cost – Row 2	············→	6.6	22	20	42
Minimize Transit Cost – Row 3	············→	5.7	19	20	39
Minimize Transit Cost – Row 4	············→	6.0	20	20	40
Minimize Transit Cost – Row 5	············→	6.9	23	20	43
Minimize Actual Total Cost (X = 1)	– – – – →	4.2	14	24	38
Weighted To Min. Cost (X = 0.75) – Rte 1	— · · —→	3.3	11	28	39
Weighted To Min. Cost (X = 0.75) – Rte 2	— · · —→	4.2	14	24	38
Evenly Weighted (X = 0.50)	— — →	3.3	11	28	39
Weighted To Min. Piracy (X = 0.25) – Rte 1	— · — →	3.3	11	28	39
Weighted To Min. Piracy (X = 0.25) – Rte 2	— · — →	3.0	10	32	42
Weighted To Min. Piracy (X = 0.25) – Rte 3	— · — →	3.0	10	32	42
Weighted To Min. Piracy (X = 0.25) – Rte 4	— · — →	2.7	9	36	45
Minimize Piracy Cost (X = 0)	———→	2.7	9	36	45

In the case of priority choices that result in multiple route options, a mariner could use a secondary preference to decide between options for that choice. For example, the choice to minimize transit cost, ignoring the potential cost related to a successful pirate attack, results in five route options, each of which have the same transit cost of $20,000. If a secondary preference of minimizing total actual cost of the route, which includes EMV_{piracy}, were considered, then the choice could be narrowed down to following row 3 of the matrix straight across.

Using a table of potential routes such as Table 6, a mariner can make a route decision that best aligns with his pirate avoidance and financial goals, while still being aware of the tradeoffs involved in such a decision. For many situations, seeing the impact of his priority choices on the probability of piracy and the actual cost of a route may lead him to following a course that is a slight compromise. For example, in the above example, after analyzing the route options, many mariners would ultimately choose to follow the Evenly Weighted (X = 0.50) route. Compared to the Piracy Avoidance (X = 0) route, the Evenly Weighted route decreases the overall cost by $6,000 while only increasing the Piracy Probability by 0.6 thousandths. Meanwhile, compared to the Minimum Actual Total Cost (X = 1) route, it only raises the overall cost by $1,000 while decreasing the Piracy Probability by 0.9 thousandths. Again, however, the selection of an Optimized Transit Path really depends on a mariner's choices regarding his priorities for piracy avoidance versus cost minimization.

Chapter 6: Survey of Deterministic Shortest Path Algorithms

6.1 Types of Deterministic Shortest Path Problems

In deterministic shortest path problems, all values in the matrix are treated as absolute, with no probability distribution associated with each node. The shortest path algorithm attempts to find the path between two nodes in the matrix such that the sum of the distances between the nodes is minimized. This is analogous to a traveler trying to find the quickest or shortest route via roads on a map from his current location to a desired destination.

There are generally three main types of deterministic shortest path problems. In single-pair shortest path problems, the shortest path from a single, defined start node to one particular end node is determined. In single-source shortest path problems, the shortest path from a single, defined start node is found to all other nodes in the matrix, typically by a series of calculations forward through the matrix. In the all-pairs shortest path problem, the shortest path between every possible pair of nodes in the graph is found (Abraham et al. 2010). Within these methods, each point in the matrix is typically referred to as a node or a vertex. The path that connects one node directly to another is referred to as an edge or a link. The distance from one node to a connected one is often referred to as the edge weight or the cost.

The Optimized Transit Path problem for maritime piracy is a hybrid of the above problem types. The piracy shortest path problem attempts to find the shortest path from any node on one far side of the matrix to any node on the opposite far side of the matrix. The piracy shortest path problem essentially allows the path to start from any point on the “starting line” (e.g. the far north edge of the Arabian Sea region) and requires the path to end at any point on the “finish line” (e.g. the far south edge of the Arabian Sea region, where the danger of encountering pirates has passed). Shortest paths between points inside the matrix are only relevant insofar as they contribute to the shortest path from one side of the matrix to the other.

6.2 Dijkstra's Algorithm

To solve the single-pair shortest path problem, Dijkstra's Algorithm is the most commonly used algorithm. For a given source node, Dijkstra's Algorithm finds the path with the lowest net cost (sum of edge weights) to get to a particular destination node. The algorithm requires that all edge weights in the matrix be non-negative. By letting it run fully instead of stopping the algorithm once the desired destination is reached, Dijkstra's Algorithm can also be used to solve the single-source shortest path problem, giving the shortest path from a defined start node to all other nodes in the matrix.

Dijkstra's Algorithm starts by labeling the value for the starting point as zero and temporarily labeling the value for every other node as infinity. It then recalculates the value for each node directly connected to the starting node as the starting node's value (zero) plus the distance from the starting node to that node. The starting node is then labeled as visited and will not enter the calculations again. Meanwhile, of all the remaining nodes, the one with the smallest value (which is the shortest distance from the starting point) becomes the current node of interest. The values are recalculated for all nodes directly connected to the current node. The values for each of these connected nodes equals the value of the current node plus the distance from the current node to each of them. The current node is then labeled as visited and does not enter into future calculations. As before, the remaining node with the smallest value becomes the new current node. The values of all nodes directly connected to it are recalculated, leading to selection of the minimum as the next current node. This process continues until every node in the matrix has been labeled as visited. At this point, the value of each node equals the length of the shortest path from the starting node to it (Winston 2004).

The worst case performance of Dijkstra's original algorithm, first conceived in 1956, requires on the order of $O(N^2)$ calculations, where N is the number of nodes in the matrix (Dijkstra 1959, Winston 2004). However, by implementing a Min-Priority Queue with a Fibonacci Heap along with Dijkstra's Algorithm, the solution can be reached on the order of $O(E + N \log N)$ calculations, where N is the number of nodes in the matrix and E is the number of edges between nodes (Fredman and Tarjan 1984, Cormen et al. 2009).

In the Min-Priority Queue with Fibonacci Heap method, the matrix is typically stored in the form of adjacency lists telling which nodes are directly connected to which. Certain values in these node heaps can be pre-calculated. Then, instead of recalculating values for almost every node in the matrix every time a new current node is considered, certain heaps are given priority as the most likely sources of the next current node. Dijkstra's Algorithm implementing Min-Priority Queue with a Fibonacci Heap is especially efficient in dispersed matrices, where all nodes are not connected to each other (Cormen et al. 2009). This is considered the fastest single-source shortest path algorithm for matrices with non-negative edge weights. Because of its speed and versatility, Dijkstra's Algorithm with Min-Priority Queue and Fibonacci Heap is widely used in network routing protocols, especially OSPF (Open Shortest Path First) and IS-IS (Intermediate System to Intermediate System), as well as most transportation routing applications (Chen et al. 2007).

For most everyday applications involving shortest path calculations, such as transportation routing problems, edge weights are typically positive. For example, distances between two locations are always non-negative. However, for certain applications, edge weights could be negative. For example, if the edge weights in a matrix represented financial outcomes, then both positive (financial gain) and negative (financial loss) outcomes would be possible. In such a case, Dijkstra's Algorithm could not be used, and other shortest path methods would have to be utilized. Some of these other options are discussed in the remainder of Chapter 6.

For the 43x50 PARS matrix used in maritime piracy prediction, there are N = 2,150 nodes and E = 8,323 edges if motion is allowed in the horizontal, vertical, or diagonal directions (e.g if motion is allowed in 8 potential directions from each node). However, in order to keep the distance traveled from one node to the next constant, it may be desired to restrict motion to only the horizontal and vertical directions (e.g. motion is allowed in only 4 potential directions from each node). In this case, there are 4,207 edges.

Thus, for the case where diagonal motion is permitted, the solution to the traditional Dijkstra's Algorithm would require $O(N^2) = (2{,}150)^2 = 4.6$ million calculations, while the Min-Priority Queue with Fibonacci Heap version would require only $O(E + N \log N) = (8{,}323 + 2{,}150 * \log (2{,}150)) = 15{,}500$ calculations, for a solution from a single node to all other nodes. Meanwhile, if diagonal motion is not permitted, the solution to the traditional Dijkstra's algorithm would still require $O(N^2) = (2{,}150)^2 = 4.6$ million calculations, while the faster version would require only $O(E + N \log N) = (4{,}207 + 2{,}150 * \log (2{,}150)) = 11{,}372$ calculations.

Because the maritime piracy application allows for starting from any of the 43 nodes on the “starting line” of the PARS matrix, Dijkstra’s Algorithm would have to be run 43 separate times, starting from each of the potential starting points, to reach a definite conclusion on the shortest path through the matrix. Thus, almost 200 million calculations would be required for the traditional algorithm. Meanwhile, only 670,000 calculations would be required for Dijkstra’s Algorithm implementing Min-Priority Queue with a Fibonacci Heap if diagonal motion is allowed. This number drops to 490,000 calculations if diagonal motion is not allowed.

6.3 Bellman-Ford Algorithm

Another method commonly used to solve the single-source shortest path problem is the Bellman-Ford Algorithm. As with Dijkstra's Algorithm, it computes the shortest path from a single source node to all other nodes in the matrix. It is generally slower than Dijkstra's Algorithm, as its solution requires on the order of O(N * E) calculations, where N is the number of nodes and E is the number of edges in the matrix. However, the Bellman-Ford Algorithm is more versatile, as it does not require that all edge weights be non-negative, though no negative-weight cycles may exist (Bellman 1958, Bang-Jensen and Gutin 2009). For the piracy application, however, all probability and cost values are positive numbers, so this benefit of the Bellman-Ford algorithm would not be utilized.

For one pass through the 43x50 PARS matrix, the Bellman-Ford Algorithm would require O(N * E) = 2,150 * 8,323 = 17.9 million calculations if diagonal motion is included. If diagonal motion is excluded, the number of required calculations drops to O(N * E) = 2,150 * 4,207 = 9.05 million.

To arrive at a full solution to the Optimized Transit Path problem for maritime piracy would require an additional 43 times through the matrix to check on all potential starting points. This would result in 770 million calculations to solve the problem if diagonal motion is included or 390 million calculations if it is not.

6.4 Floyd-Warshall Algorithm and Johnson's Algorithm

To solve the all-pairs shortest path problem, which compares all possible shortest paths through the matrix between every pair of nodes, the Floyd-Warshall Algorithm and Johnson's Algorithm are commonly used. Both algorithms are able to handle both positive and negative edge weights, though no negative-weight cycles may exist. These algorithms are probably not appropriate to quick solution of the piracy Optimized Transit Path problem, as they add significant capabilities, at the cost of extra calculations, not needed in the piracy problem. Both algorithms solve for the shortest path between all possible combinations of points. The piracy problem, however, only requires the shortest path between points on the "starting line" at one end of the matrix and those on the "finish line" at the other side. All of the calculations of shortest paths in the middle of the matrix would be superfluous and unnecessary for the piracy shortest path application. Also, both algorithms allow negative edge weights, which leads to extra layers of solution complexity. However, in the piracy problem, all probabilities and costs are positive, so there are no negative edge weights. Thus, this capability goes unused.

The solution to the Floyd-Warshall Algorithm requires on the order of $O(N^3)$ calculations (Floyd 1962, Rosen 2012). Johnson's Algorithm requires on the order of $O(N^2 \log N + N * E)$ calculations (Johnson 1977, Black 2004). In the case of sparser matrices with relatively few edges, Johnson's Algorithm may be a faster method of solving for shortest path. For the 43x50 PARS matrix with diagonal motion permitted, the Floyd-Warshall Algorithm requires $O(N^3) = (2{,}150)^3 = 994$ billion calculations, while Johnson's Algorithm would require $O(N^2 \log N + N * E) = ((2{,}150)^2 \log (2{,}150)$

+ 2,150 * 8,323) = 33 million calculations. If diagonal motion is not allowed, the Floyd-Warshall algorithm still requires 994 billion calculations, while the number for Johnson's Algorithm drops to O(N^2 log N + N * E) = ($(2,150)^2$ log (2,150) + 2,150 * 4,207) = 24.5 million calculations.

Chapter 7: The Optimized Transit Path (OTP) Algorithm

In order to efficiently solve for a mariner's "best" transit path through the Arabian Sea, one that allows him to both avoid pirates and minimize costs, the Optimized Transit Path (OTP) algorithm was developed. This algorithm quickly solves for the shortest path through a matrix from any point on one side of the matrix, the "starting line," to any point on the opposite side, the "finish line." Though developed for the piracy problem, it is appropriate for finding the shortest path in many applications involving two-dimensional movement on a flat geographic plane.

Unlike traditional shortest path algorithms, the Optimized Transit Path algorithm treats path minimization as more of a "scheduling" problem, rather than an Operations Research optimization problem. The Critical Path Method (CPM), used in project scheduling, identifies the maximum length path through a network. In a like manner, the OTP algorithm uses similar approaches to find the minimum length path through a matrix. This "scheduling" approach to the shortest path problem provides insights not gained from classical shortest path approaches. Specifically, the Optimized Transit Path algorithm identifies the additional cost associated with diverting from the shortest path, information not provided by traditional methods.

The Optimized Transit Path algorithm can be set up to allow motion between nodes in the up/down, left/right, or diagonal directions (8 possible movements from each node). Or, it can easily be adapted to restrict motion between nodes to only up/down and left/right directions (4 possible movements from each node). In developing the OTP algorithm, relatively simple examples, for which the solution is readily apparent, were first used to test if the logical relationships led to the "obvious" answer. The following sections describe the terminology, logic and calculations of the Optimized Transit Path algorithm, using a fairly simple example for illustration.

7.1 Background and Definitions

The Optimized Transit Path algorithm uses a methodology analogous to the forward pass and backward pass scheduling techniques of the Critical Path Method (CPM). However, while CPM tries to determine the maximum length path through a matrix (usually a schedule), the goal of the Optimized Transit Path algorithm is to find the minimum length path through a matrix (for the case of Arabian Sea piracy, an array of probability values or of costs). In addition, while CPM's use of precedence relationships keeps a path always moving forward, the Optimized Transit Path algorithm allows vertical or even backwards motion through a matrix, in addition to forwards, to achieve the minimum length path.

7.1.1 Node Matrix

For the application in this study, the Optimized Transit Path algorithm will determine the best path to follow through the Arabian Sea to minimize one's probability of encountering pirates, minimize the overall costs associated with the transit, or some combination thereof. The entire Arabian Sea region, which is bound by land masses on three sides, covers roughly 2,150 miles by 2,500 miles. The PARS model for piracy prediction divides the Arabian Sea into a 43x50 array of equally sized geographic areas. Each value in the PARS model represents the probability of encountering a pirate vessel within a 50 mile by 50 mile box at a given time. As discussed in Chapter 5, these probability values, along with information about the ship and its defenses, can be used to calculate an Expected Monetary Value of piracy cost for each node. This cost can be combined with the transit cost per node to give an overall cost per node.

Alternatively, the piracy cost and transit cost can be weighted, according to the mariner's priorities, to give a weighted cost.

A 5x5 array of probability values from the PARS model, representing piracy prediction over a 250 mile by 250 mile area, might look like Figure 24. Each node of the matrix could represent any scalar value (e.g. a probability, a cost, or a risk weight). The methodology of the Optimized Transit Path algorithm is the same regardless of what the nodes represent.

15	10	1	2	5
13	2	4	9	3
10	2	4	10	7
7	4	2	8	12
2	1	5	8	14

Figure 24: Example 5x5 Matrix of Piracy Prediction Values

For the piracy prediction matrix, whether of probability or cost, each value in the matrix is representative of a 50 mile by 50 mile box of area in the Arabian Sea. In drawing the matrix, each value can be represented by a box, as is done with duration values in Activity In the Box CPM methods. For example, the 5x5 matrix of probability values in Figure 24 could be represented with boxes, as shown in Figure 25.

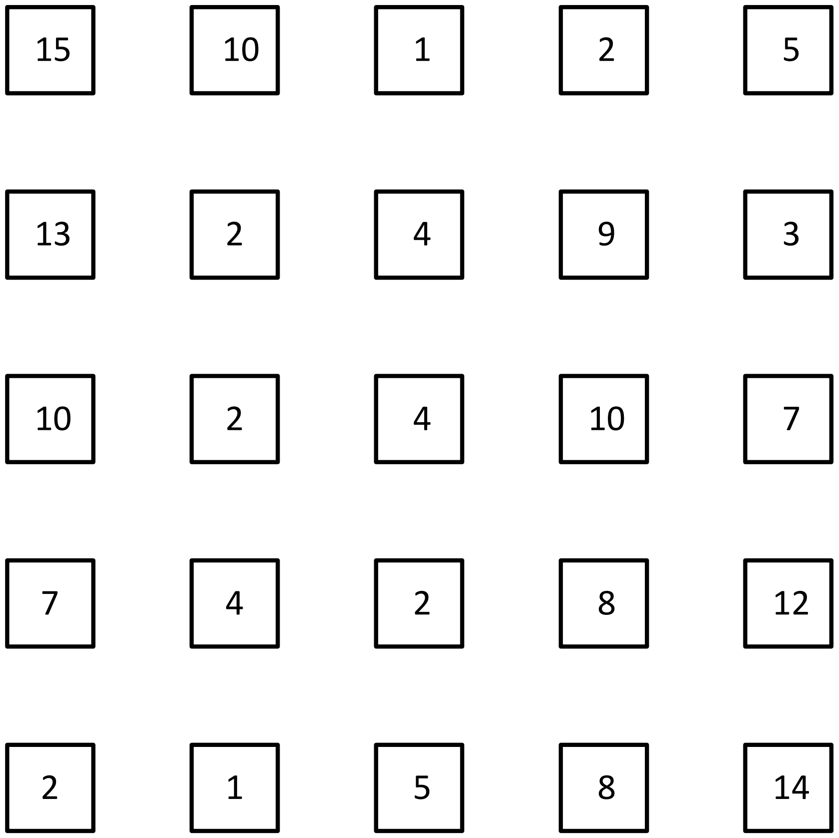


Figure 25: Example 5x5 Matrix Represented in Box Format

For the piracy application, the values in each box represent either the probability of encountering a pirate or an associated cost within that 50 mile by 50 mile region at a given time. In a map representation, the boxes would be touching. In the matrix representation, the boxes are separated in order to show the potential transit paths from one box to the next. From a box within the middle of the matrix, one could travel in any of eight directions to another box, as highlighted in Figure 26.

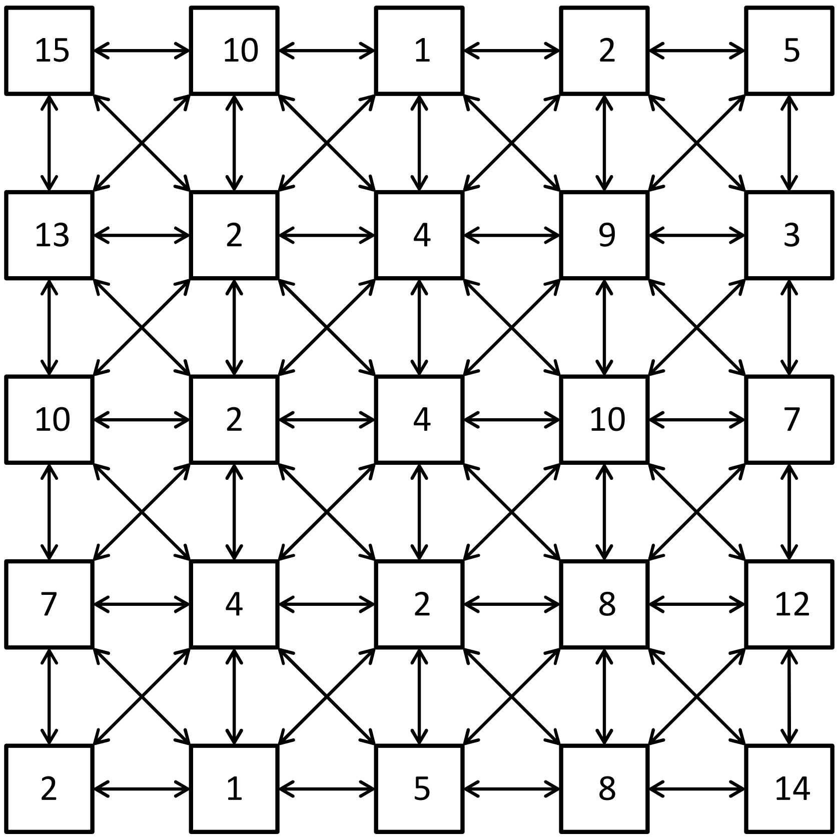


Figure 26: Example 5x5 Matrix with Potential Transit Paths Shown

For the piracy application, each of the boxes represents a 50 mile by 50 mile square area. In order to limit travel only to equal distances between nodes, motion between nodes can be restricted to only the four directions of up/down and right/left. This is especially valuable for calculations involving transit cost, which is related to distance traveled, in order to keep the transit cost associated with movement between any two nodes equivalent. However, for demonstrating the Optimized Transit Path algorithm, motion will be permitted in all eight directions from any node. Only minor changes are needed in order to restrict motion to the four equidistant directions. These changes will be discussed in section 7.5.

For illustrative purposes in this paper, a 5x5 array will be used to demonstrate the Optimized Transit Path methodology, rather than the full 43x50 matrix of the Arabian Sea from the PARS model. In addition, for simplicity and clarity of graphics, the transit path arrows between boxes will be left off in future graphics. However, it should be noted that, for this demonstration of the algorithm, travel is possible in any of the eight directions between adjoining boxes.

7.1.2 Cost of Each Node

Within each box in the node field, a cost value is indicated. In analysis of Arabian Sea piracy, this value could represent the probability, at a given moment in time, that pirates will be present somewhere within the geographic region represented by that box. These probability values come from the PARS prediction model for pirate activity. Alternatively, the cost value in each node could represent a monetary cost associated with traveling through the area represented by that node. As discussed in Chapter 5, this cost could equal the Expected Monetary Value cost of piracy in that node, the transit cost for that node, or a weighted combination of the two.

7.1.3 Node Identifier

The unique identifier for each node is its (row number, column number) coordinates in the field. For example, in Figure 26 on the previous page, node (4,3) is in the 4th row down and is the 3rd block from the left. It has a cost value of 2.

7.1.4 Minimum Start (MS) and Minimum Finish (MF) of Each Node

Ultimately, the goal of this algorithm is to determine the Optimized Transit Path through the entire matrix, with the lowest net sum of node costs from one side of the field to the other. In the piracy application, this represents the path that a mariner should navigate through the Arabian Sea to minimize his chances of encountering pirates or to minimize total costs associated with the route.

To determine the Optimized Transit Path, the Minimum Start (MS) and Minimum Finish (MF) for each node are found. Minimum Start (MS) is the lowest net sum of costs of nodes to follow in order to get to that point, starting from the left (e.g. the smallest net cost path to get from the starting line to that node). Minimum Finish (MF) is a node's Minimum Start plus the cost of that node (e.g. the smallest net cost to get from the starting line all the way through that node). MS, MF, and Cost for each node are graphically represented as shown in Figure 27.

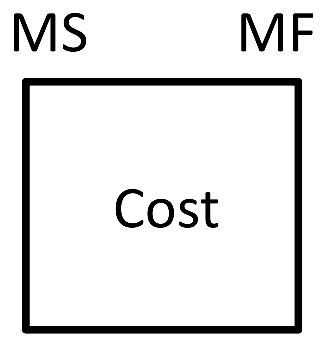


Figure 27: Method of Designating Minimum Start, Minimum Finish, and Cost

7.2 Forward Pass Calculations: Determining MS and MF for Each Node

To determine the Minimum Start (MS) and Minimum Finish (MF) values for each node, which will ultimately lead to the Optimized Transit Path through the array, a series of calculation passes are made forward through the matrix.

7.2.1 First Forward Pass

Starting from the left column of the matrix, initial values of MS and MF are calculated for each node. First, the MS value for each node in the left-most column is set at zero, 0. The MF value for each node in the left-most column is then calculated by adding the node's cost to its MS value.

$$\text{MF} = \text{MS} + \text{Cost} \tag{21}$$

For nodes in the second column, the MS value for each node is the **minimum of the MF values** of adjoining nodes from the previous column. For most of the nodes, there are 3 adjoining nodes in the previous column: the node directly to its left, the node to its left and up, and the node to its left and down. For the topmost and bottommost nodes in each column, there are only 2 adjoining nodes from the previous column. Once the MS values for each node are determined, MF values are calculated by adding each node's cost to its MS value.

Following the same process and moving from left to right, the initial MS and MF values for each node in the matrix can be calculated. For the example 5x5 array introduced in Figure 24 through Figure 26, the first forward pass gives MS and MF values for each node, as shown in Figure 28. Note that, during the first forward pass, only MF values of the adjacent nodes in the column to a node's left are used in determining its MS value. This is done in order to initially populate the matrix with MS and MF values. In subsequent forward passes, to be discussed in Sections 7.2.2 through 7.2.4, MF values from all eight surrounding nodes will be used to determine a node's new MS value.

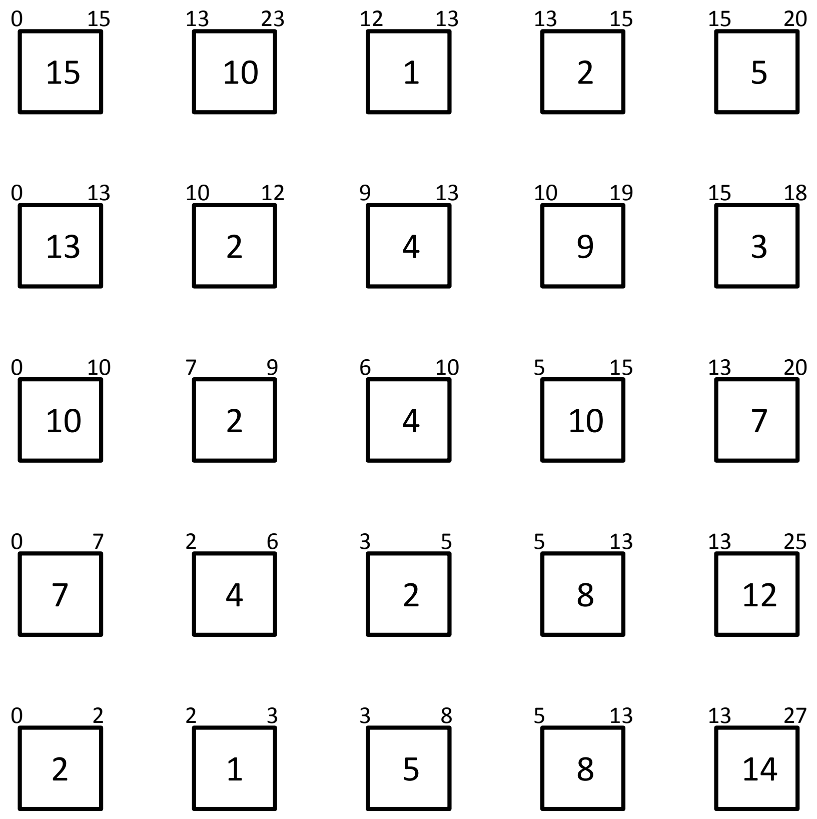


Figure 28: First Forward Pass MS and MF Values for Example 5x5 Matrix

7.2.2 Second Forward Pass

The first forward pass usually does not reveal the Optimized Transit Path because it only allows forward motion through the array. The first forward pass did not allow motion straight up or down, nor did it allow backtracking. The second forward pass through the array introduces the ability to go in all directions and moves the MS and MF values closer to their optimized minimum values.

In the second forward pass, each node begins with the MS and MF values it had at the end of the first forward pass. The first column remains identical to the first forward pass, since the Minimum Start time for each node in the column could never be less than 0. Likewise, the Minimum Finish values for all nodes in the first column remain the same, the Minimum Start (0) plus the node's cost.

In the second column of the array, MS values for each node are updated. For each node in the column, its MS value will be the **minimum** of the MF values of the 8 other nodes that adjoin it (or 5 adjacent nodes for the topmost and bottommost nodes in the column). The MS value for each node is the **minimum** of:

1. The MF values of adjoining nodes from the column to its left. These are MF values that were just updated during consideration of that column in this forward pass.
2. The MF values of nodes immediately above and below it. These are MF values of nodes from the previous forward pass, as MF values in this column have not yet been updated.

3. The MF values of adjoining nodes from the column to its right. These are also MF values of nodes from the previous forward pass, as these MF values have not been updated yet either.

Once all of the new MS values for each node in a column have been determined, based on the minimum of adjacent MF values, a new MF value is calculated for each node by adding the new MS value plus the node's cost. Depending on the arrangement of the array, numerous nodes might change their MS and MF values from the previous forward pass, or none might change.

The third column of the array is updated in a like manner to the second. The new MS value for each node is determined by finding the minimum MF value of the 8 (or 5) nodes immediately surrounding it. The new MS value for each node will be either the minimum MF from the adjacent previous (left) column rows calculated in that same forward pass, the minimum MF from the nodes directly above and below it calculated in the previous forward pass, or the minimum MF from the adjacent next (right) column nodes calculated in the previous forward pass. Once all MS values are updated in the third column, MF values for each node are recalculated.

In a like manner, MS and MF values for all nodes in the fourth, fifth, and all subsequent columns are calculated. The process goes systematically from the left column to the right column in the array, updating MS and MF values in each column before proceeding to the next.

If, upon updating MS and MF values in all columns, no changes were made to any values in any columns, then the solution has converged, the path through the array has been optimized, and no further updates are needed. If, however, changes were made to any MS or MF values in the matrix compared to the previous forward pass, then an additional forward pass is needed in order to move further towards optimization. Additional forward passes will continue until no more changes are made from one forward pass to the next, at which point the solution will have converged. Required computation time for the OTP algorithm to reach convergence will be derived later in Sections 9.3 and 9.4, as well as compared to traditional shortest path algorithms in Section 9.5.

To make the process of updating MS and MF values more clear, the second forward pass through the example 5x5 matrix is illustrated. The columns are updated one at a time, from left to right, until the entire array has been updated.

For the example 5x5 matrix (or any array), MS and MF values in the left-most column of the array never change compared to the first forward pass. In the second column, however, changes may occur during the second forward pass (Figure 29).

For node (1,2), the MS value from the first pass was 13. The MF values of adjacent Column 1 nodes from the second pass are 15 and 13. The MF value of the node directly below it from the first pass is 12. And the MF values of the Column 3 nodes immediately after it from the first pass are 13 and 13. The minimum of these MF values is 12. Thus, the MS value of node (1,2) changes to 12. For clarity on the graphics for Figure 29, the updated MS value was boxed, as was the MF value that dictated this change.

For node (2,2), the MS value from the first pass was 10. The MF values of adjacent Column 1 nodes from the second pass are 15, 13, and 10. The MF values of the nodes directly above and below it from the first pass are 23 and 9. And the MF values of

Column 3 nodes immediately after it from the first pass are 13, 13, and 10. The minimum of these MF values is 9, which is less than the node's starting MS value. Thus, the MS value of node (2,2) changes to 9. For clarity on the graphics for Figure 29, the new MS and the originating MF value were circled.

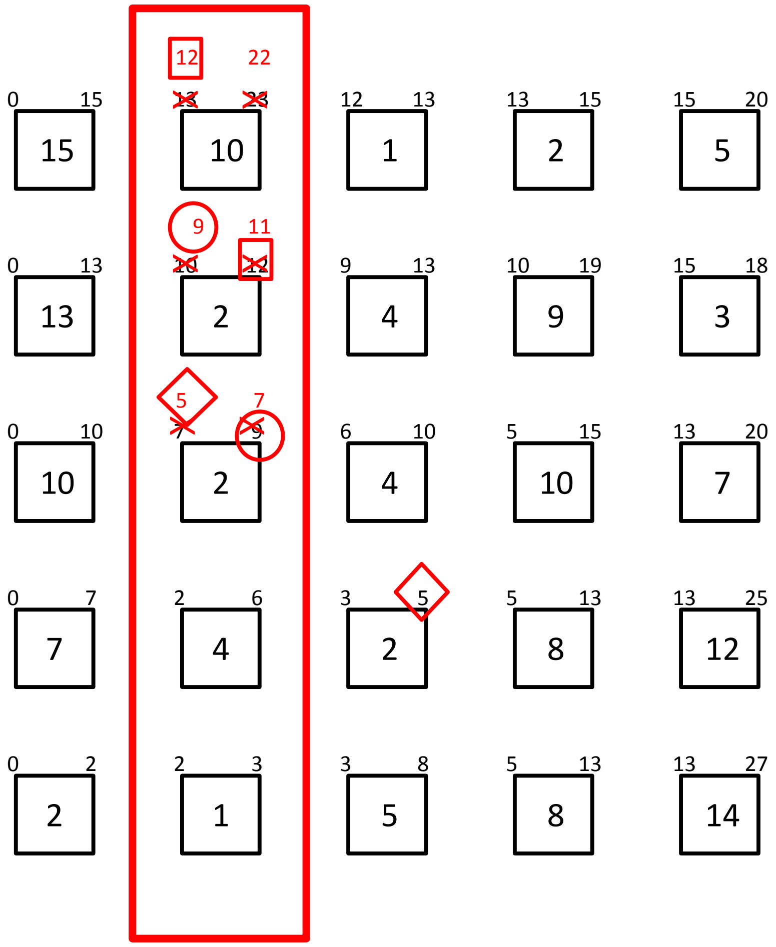


Figure 29: Second Forward Pass - Column 2 of Example 5x5 Matrix

In a like manner, the MS value for node (3,2) changes from 7 to 5. This is because the MF value of the node immediately after and below it (node (4,3)) in the previous pass was 5. Both the new MS value and the MF value that led to the change are highlighted by diamonds in Figure 29.

Following the same methodology, neither node (4,2) nor node (5,2) have changes to their MS values during the second forward pass. Once all of the new MS values for the second column have been determined, the new MF value for each node in the column is calculated by adding the node's cost to its MS value.

Next, the third column is updated for the second forward pass (Figure 30). Note that the second forward pass values for Column 2 are used when calculating changes in Column 3.

Node (1,3) previously had an MS value of 12. From the just-updated column before it, MF values of 22 and 11 adjoin it. From the previous forward pass, MF values of 13, 15, and 19 surround the node below and after it. The minimum of these MF values is 11, so the MS value of node (1,3) changes to 11. Both the new MS value and the MF value that dictated the change are circled in Figure 30.

Likewise, node (2,3), which previously had an MS value of 9, is surrounded by MF values of 22, 11, and 7 before it (from the second pass calculations), MF values of 13 and 10 above and below it (from the first pass calculations), and MF values of 15, 19, and 15 after it (from the first pass calculations). The minimum of these is 7, which becomes the new value of MS for node (2,3). This change is highlighted by diamonds in Figure 30

In a like manner, the MS value of node (3,3) is reduced from 6 to 5 (due to the MF values of node (4,3) below it). This change is highlighted by triangles in Figure 30.

Following this methodology, the MS values of nodes (4,3) and (5,3) do not change during the second forward pass.

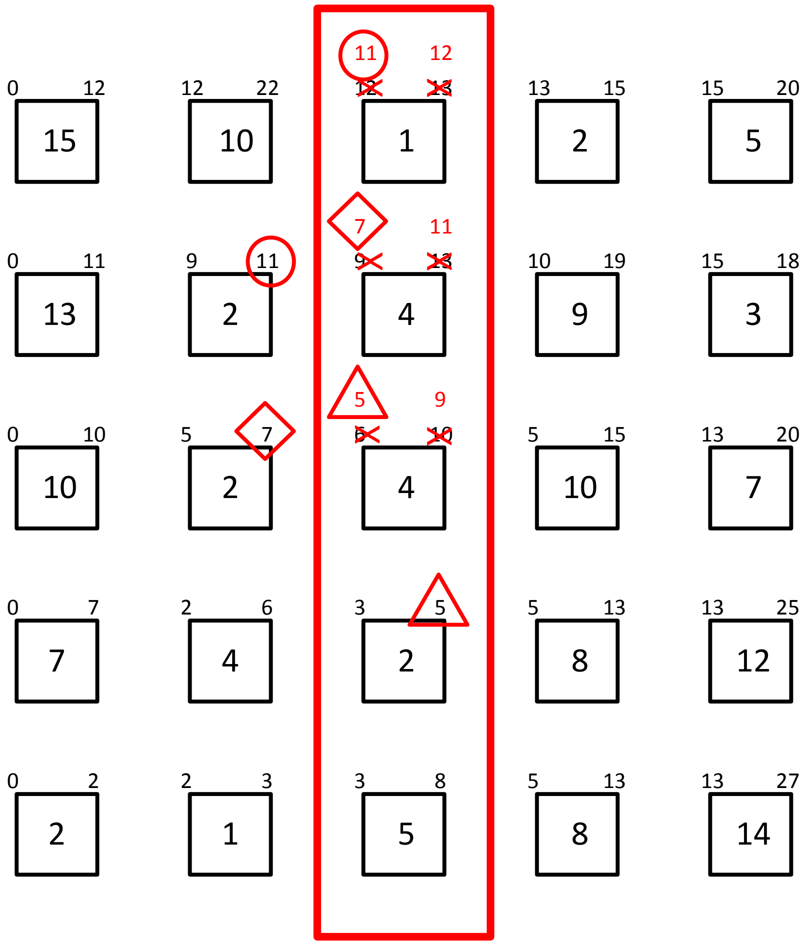


Figure 30: Second Forward Pass - Column 3 of Example 5x5 Matrix

Once all of the new MS values for the third column have been determined, the new MF value for each node in the column is calculated and updates begin on MS values for the fourth column.

Calculations for the fourth column proceed in a like manner. For the example 5x5 matrix, only nodes (1,4) and (2,4) experience changes, which are highlighted by circles and diamonds, respectively, in Figure 31.

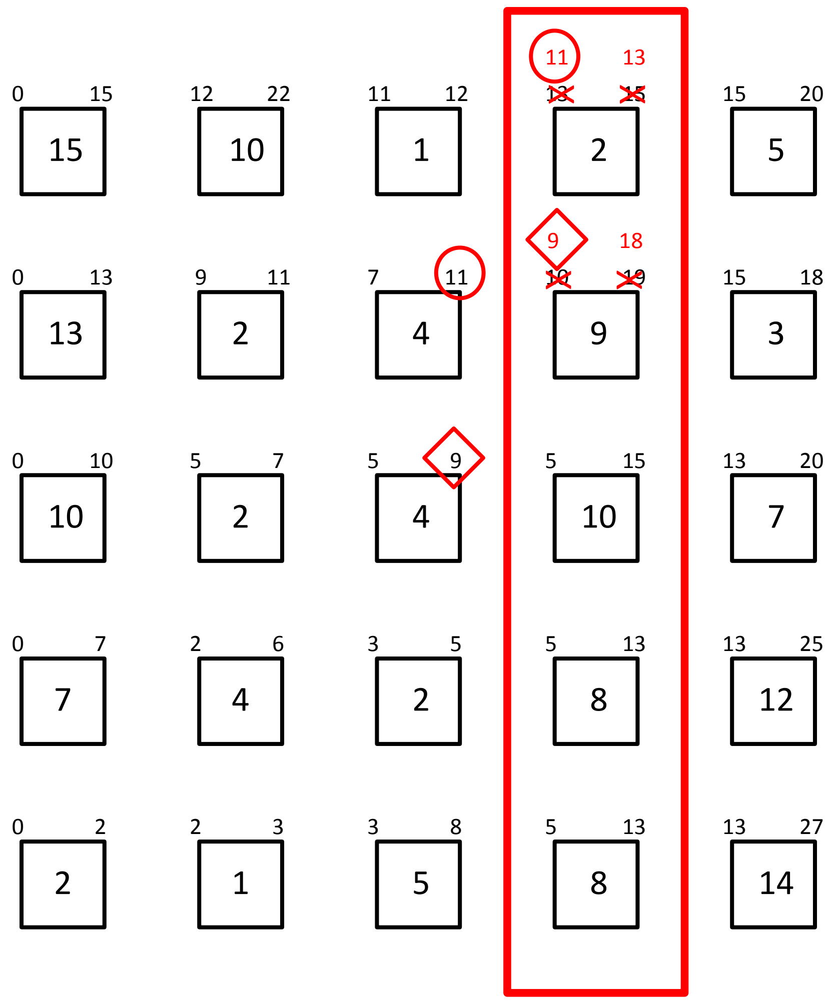


Figure 31: Second Forward Pass - Column 4 of Example 5x5 Matrix

The process continues methodically from left to right until all columns have been updated for the entire array. For the example 5x5 matrix, nodes (1,5) and (2,5) change during second pass calculations of the fifth, and final, column (Figure 32).

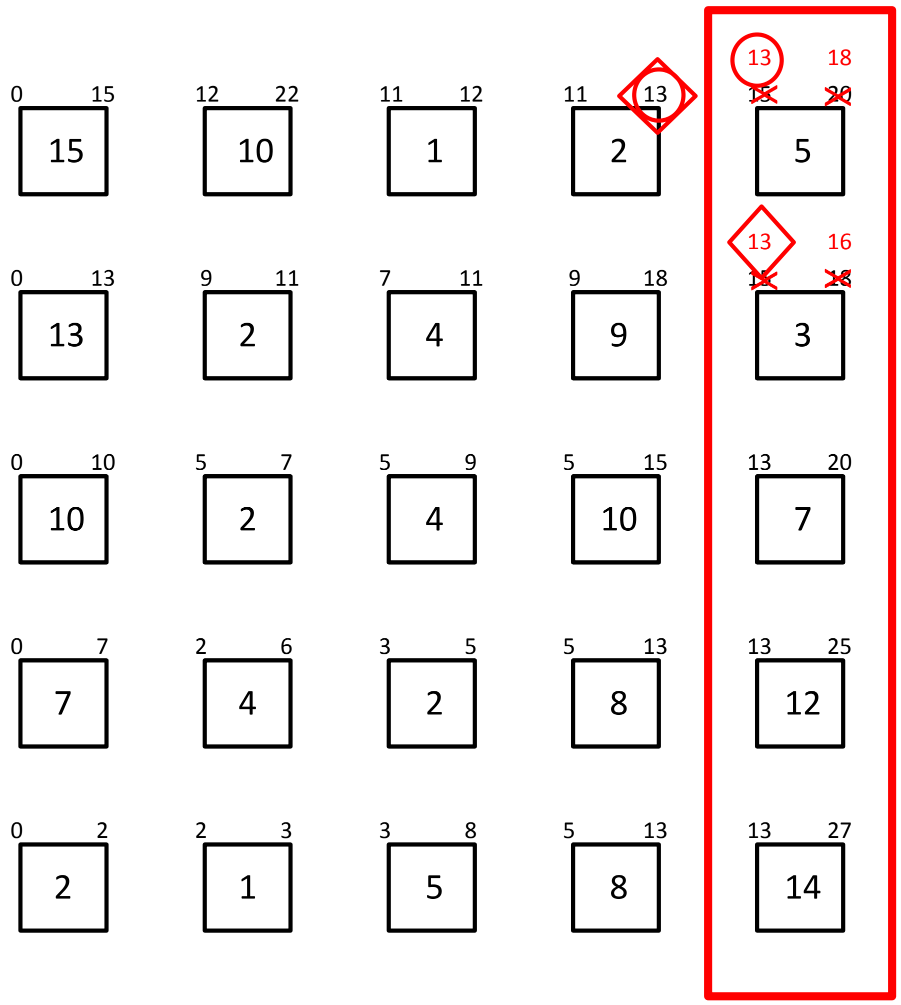


Figure 32: Second Forward Pass - Column 5 of Example 5x5 Matrix

If no changes were made to any MS or MF values in the entire matrix during the second pass, then the solution to the array would be optimized. If changes were made, as they were in the above example within Columns 2, 3, 4, and 5, then an additional forward pass is needed to move further towards an optimized solution.

7.2.3 Additional Forward Passes

The same methodology is used for the third forward pass through the array, with results of the second forward pass providing the starting MS and MF values for each node. The MS value for each node in a column is updated based on MF values of the 8 (or 5) nodes adjoining it. Once all the nodes in a column are updated with new MS and MF values, values for the next column are determined. Updating of the array's MS and MF values proceeds from the left-most column to the right, with one column at a time updated. For the example 5x5 matrix, the results of the third forward pass are shown in Figure 33.

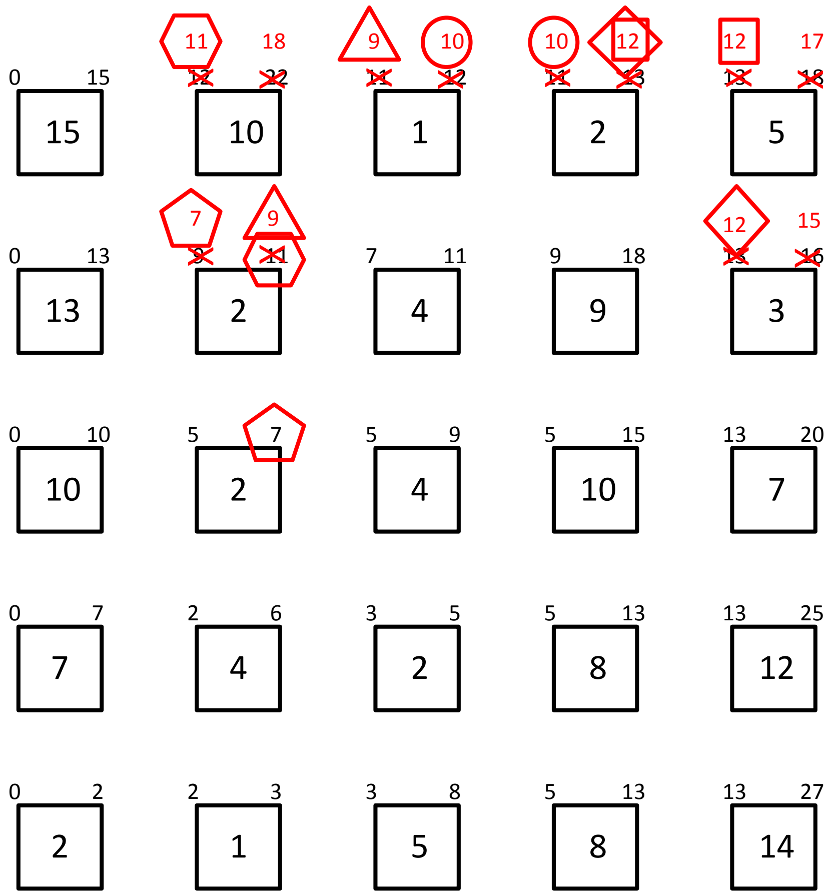


Figure 33: Third Forward Pass MS and MF Values for Example 5x5 Matrix

Because MS and MF values were changed during the third forward pass, another forward pass is needed. The fourth forward pass through the example 5x5 matrix results in changes to only one node (Figure 34).

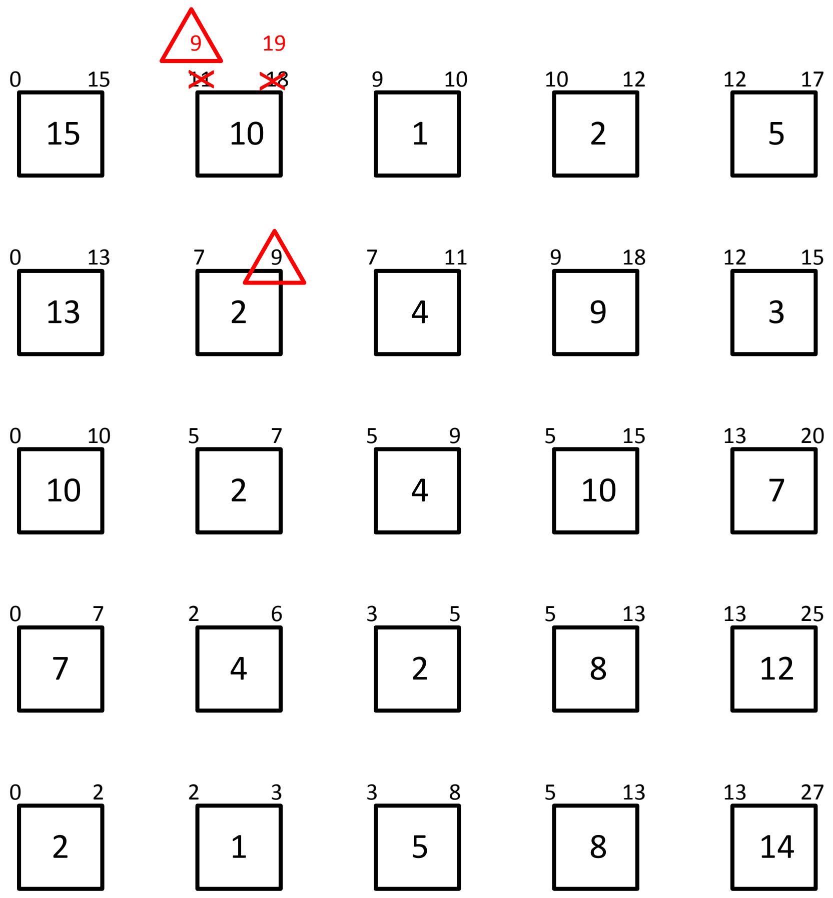


Figure 34: Fourth Forward Pass MS and MF Values for Example 5x5 Matrix

Only one node had a change in its MS and MF values compared to the previous forward pass. However, another forward pass is needed to confirm whether or not the Optimized Transit Path has been determined (Figure 35).

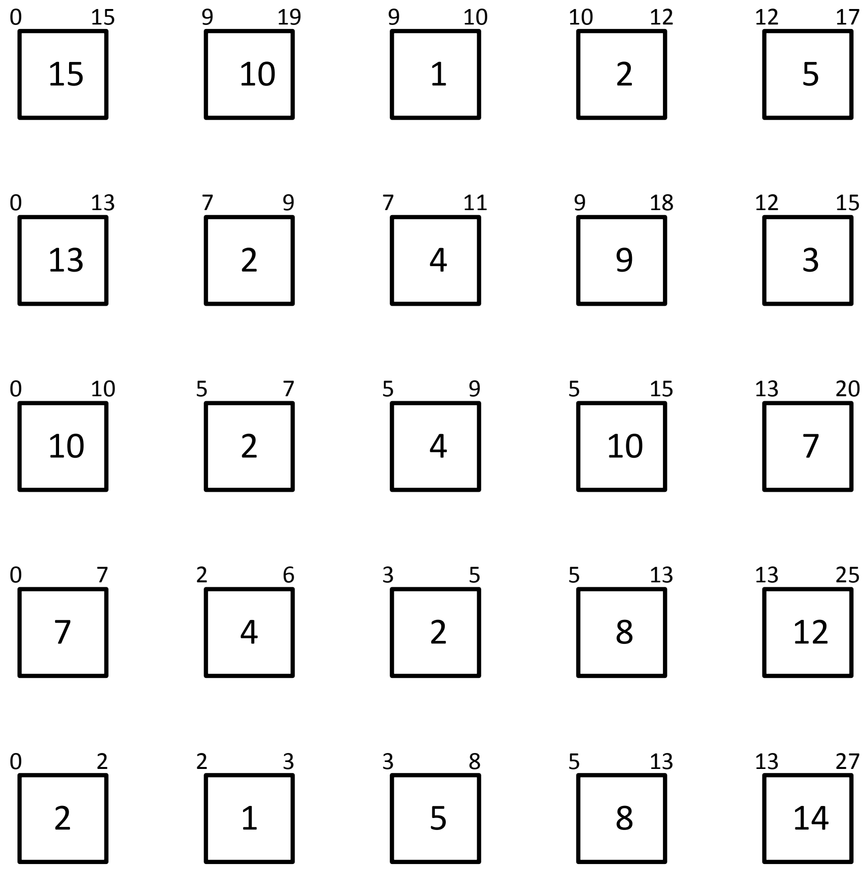


Figure 35: Fifth Forward Pass MS and MF Values for Example 5x5 Matrix

7.2.4 Final Forward Pass: Optimized Transit Path through the Matrix

For the example 5x5 matrix, the results for the fifth forward pass are identical to those for the fourth. This means that MS and MF values for all nodes in the array have been optimized for their minimum possible values. For each node in the matrix, the minimum possible net cost to travel to that node, starting from the left edge of the array, is its Minimum Start (MS) value. The smallest of the Minimum Finish (MF) values of nodes in the right-most column is the minimum net cost to pass through the entire matrix.

For the maritime piracy application, the meaning of MS and MF values depends on what the cost of each node represents. If node cost represents the probability of encountering pirates in that 50 mile by 50 mile area, then the MS value for each node is the minimum net probability of encountering pirates when traveling to that node's location. The minimum MF value of all nodes in the right-most column represents the net probability of encountering pirates if the Optimized Transit Path is followed. The Optimized Transit Path would be the path through the array with the smallest net probability of pirate activity.

Meanwhile, if the cost of each node represents an actual cost (whether an overall cost, a weighted cost, or a risk cost), MS of each node is the minimum cost to get to that node. The minimum MF of nodes is the right-most column is the minimum cost of traveling through the matrix by following the Optimized Transit Path. In this case, the Optimized Transit Path is the route through the matrix with the smallest total cost.

7.2.5 One Process to Identify the Optimized Transit Path Through the Matrix

Once the Minimum Start (MS) and Minimum Finish (MF) value have been determined for each node in the matrix, the Optimized Transit Path can be identified by starting at the finish node with the lowest MF, then drawing a path that connects it to the node that determined its MS value (the adjacent node with the lowest MF). Likewise, that node is connected to the adjacent node that determined its MS value. The process of linking nodes continues until the starting column is reached, thus revealing the Optimized Transit Path through the matrix. For the example 5x5 matrix of Figure 35, for which the minimum MS and MF values have been determined for all nodes, the Optimized Transit Path through the matrix is shown in Figure 36.

However, this manual method of determining the Optimized Transit Path can be cumbersome and difficult, especially for a large matrix or one with lots of similar values in it. In addition, this method gives no indication of the penalties (increased net cost) that would be associated with diverting from the Optimized Transit Path. In the maritime piracy example, such information could be valuable. For example, if the Optimized Transit Path for lowest net piracy probability required traveling twice as far in physical distance as another path that had only slightly lower net probability, then a mariner might choose to follow the slightly riskier, but shorter, path in order to save on fuel costs and travel time.

Another method is needed in order to better identify the Optimized Transit Path and determine the penalty for veering from it.

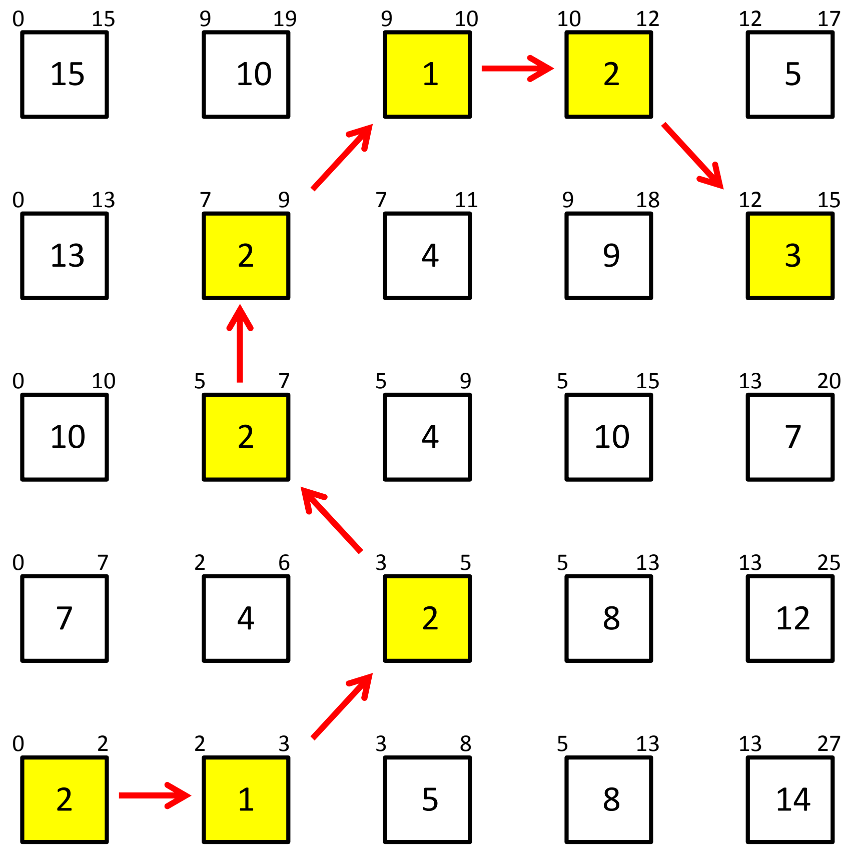


Figure 36: Optimized Transit Path Forward through Example 5x5 Matrix

7.3 Backward Pass Calculations: Determining RS and RF for Each Node

7.3.1 Definitions: Required Finish, Required Start, and Optimized Slack

In order to more easily identify the Optimized Transit Path and quantify the penalty of diverting from it, three additional terms are determined for each node: Required Start (RS), Required Finish (RF), and Optimized Slack (OS).

Required Finish (RF) is the maximum net cost allowed for being completely through a node, if that node is going to be included in a path that reaches the end of the array within the optimized minimum total duration, which was found in the final forward pass through the array.

Required Start (RS) is a node's Required Finish minus its cost. Required Start represents the largest net cost for arriving at a node, if that node is to be part of a path that finishes within the optimized minimum total cost.

Optimized Slack (OS) of a node equals the difference between its Required Start and Minimum Start, or between its Required Finish and Minimum Finish. Optimized Slack for a node represents the additional cost that will be added to the overall total cost of the Optimized Transit Path if that node is diverted to. Negative values of Optimized Slack correspond to increases in the total minimum path duration. Optimized Slack can be used to quickly identify the Optimized Transit Path through a matrix, as will be discussed in Section 7.4.

Required Start, Required Finish, and Optimized Slack, along with terms already defined, are graphically represented for each box via the designation shown in Figure 37.

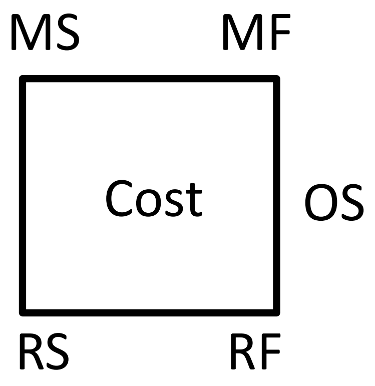


Figure 37: Method of Designating Required Start, Required Finish, and Optimized Slack

7.3.2 First Backward Pass

In order to determine the Required Start (RS) and Required Finish (RF) values for each node, which will be used to clearly identify the Optimized Transit Path through the matrix, a series of calculation passes are made backwards through the array.

Starting from the right column of the matrix, initial values of RS and RF are calculated for each node. First, the RF value for every node in the right-most column is set equal to the minimum MF value for the entire column. This value represents the minimum net cost associated with following the Optimized Transit Path through the matrix. The RS value for each node in the right-most column is then calculated by subtracting the node's cost from its RF value.

$$\text{RS} = \text{RF} - \text{Cost} \tag{22}$$

For nodes in the second column from the right, the RF value for each node is the **maximum of the RS values** of the adjacent nodes in the column to its right. For most of the nodes, there are 3 adjacent nodes in the column to its right: the node directly to its right, the node to its right and up, and the node to its right and down. For the topmost and bottommost nodes in each column, there are only 2 adjacent nodes from the column to its right. Once the RF value for each node is determined, the RS values for each node are calculated by subtracting the node's cost from its RF value.

Following the same process and moving from right to left, the initial RS and RF values for each node in the matrix are calculated.

For the example 5x5 matrix introduced earlier, the first backward pass gives RS and RF values for each node as shown in Figure 38. Note that, for the sake of clarity and instruction, the MS and MF values for each node are not shown. Also, note that the initial RF value for each node in the right-most column is set equal to 15. This equals the minimum MF value for the nodes in the column and the minimum overall cost of the Optimized Transit Path, as calculated during the forward passes.

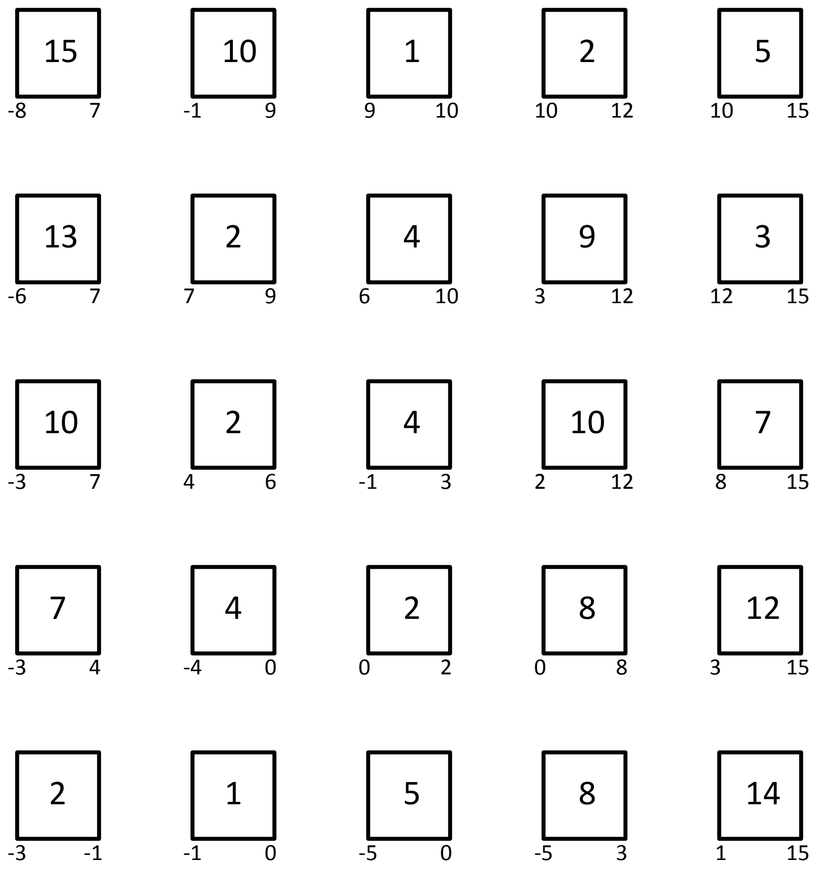


Figure 38: First Backward Pass RS and RF Values for Example 5x5 Matrix

7.3.3 Second Backward Pass

The first backward pass usually will not reveal the Optimized Transit Path because it only allows continuous motion backward through the matrix (e.g. only motion from right to left is permitted). The first backward pass did not allow motion straight up or down, nor did it allow backtracking (e.g. motion from left to right). The second backward pass through the array introduces the ability to go in all directions and moves the RS and RF values closer to their optimized values.

In the second backward pass, each node begins with the RS and RF values it had at the end of the first backward pass. The final (right-most) column remains identical to the first backward pass. The Required Finish for each node in the last column equals the minimum MF value from that column. Likewise, the Required Start for all nodes in the final column remain the same: Required Finish minus the node's cost.

For nodes in the second column from the right, RF values for each node are updated. For each node in the column, its RF value will be the maximum of the RS values of the 8 other nodes that adjoin it (or 5 adjacent nodes for the topmost and bottommost nodes in the column). The RF value for each node is the <u>**maximum**</u> of the following.

1. RS values of nodes adjoining it from the column to its right. These are the RS values that were just updated during consideration of that column in this backward pass.

2. RS values of nodes immediately above and below it. These RS values are from the previous backward pass, as RS values in this column have not yet been updated.
3. RS values of nodes adjoining it from the column to its left. These are also RS values from the previous backward pass, as these RS values have not been updated yet either.

Once all of the new RF values for each node in a column have been determined, based on the maximum of the RS values adjacent to it, a new RS value is calculated for each node by subtracting the new RF value minus the node's cost. Depending on the arrangement of the matrix, numerous nodes might change their RF and RS values from the previous backward pass, or none might change.

Nodes in the third column from the right are updated in a like manner to the second column from the right. The new RF value for each node is determined by finding the maximum RS value of the 8 (or 5) nodes immediately surrounding it. The new RF value for each node will be either the maximum RS from the adjacent previous (right) column nodes calculated in that same backward pass, the maximum RS from the nodes directly above and below it calculated in the previous backward pass, or the maximum RS from the adjacent next (left) column nodes calculated in the previous backward pass. Once all RF values are updated in the third column from the right, RS values for each node are recalculated by subtracting the node's cost from its new RF value.

In a like manner, RF and RS values for all nodes in the fourth, fifth, and all subsequent columns from the right are calculated. The process goes systematically from the right column to the left column in the matrix, updating RF and RS values in each column before proceeding to the next.

If, upon updating RF and RS values in all columns, no changes were made to any values, then the path backward through the array has been optimized and no further updates are needed. If, however, changes were made to any RF or RS values compared to the previous backward pass, then an additional backward pass is needed to move further towards optimization.

The process of updating RF and RS values while passing backward through the array may be made clearer by looking at an example. Figure 39 shows the results of the second backward pass through the example 5x5 matrix.

The final, right-most column of the array (column 5) does not change compared to the first backward pass.

In this example, no changes are necessary in the second column from the right (column 4) on the second backward pass either. For node (1,4), the RF value from the first pass was 12. The RS values of nodes adjoining it from the column to its right are still 10 and 12 on the second backward pass. Likewise, the RS value of the node directly below it from the first pass is 3. The RS values of nodes immediately to its left from the first pass are 9 and 6. The maximum of these RS values is 12, the same as the node's starting RF value. Thus, the RF value of node (1,4) does not change.

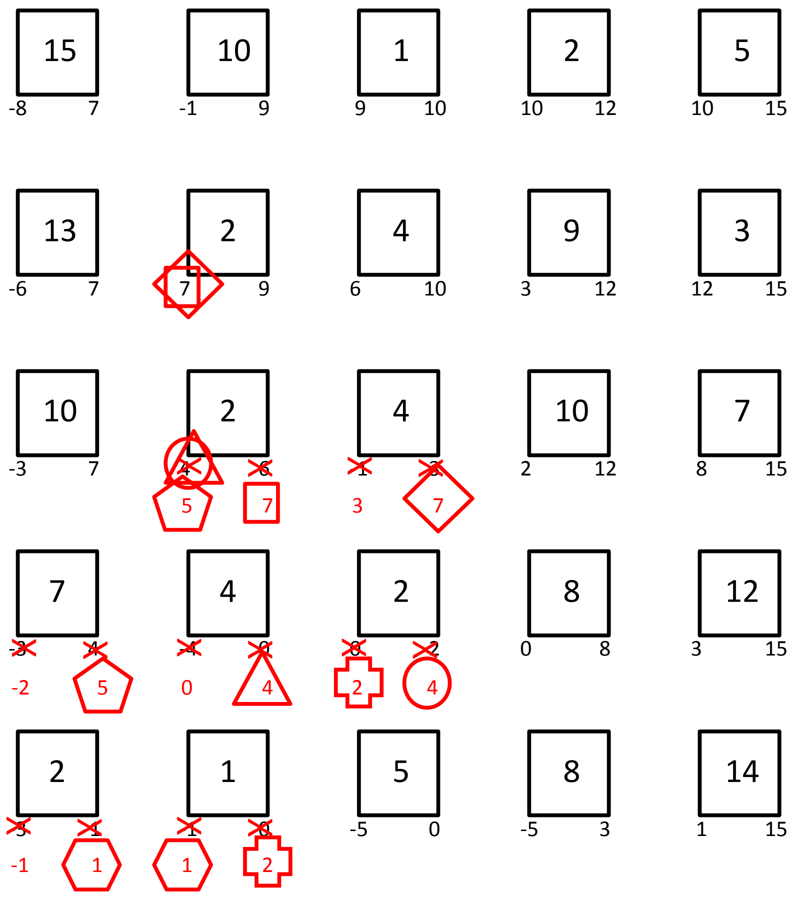


Figure 39: Second Backward Pass RF and RS Values for Example 5x5 Matrix

For node (2,4), the RF value from the first backward pass is 12. The RS values of nodes adjoining it from the column to its right from the second pass are 10, 12, and 8. The RS values of the nodes directly above and below it from the first backward pass are 10 and 2. The RS values of the nodes immediately to its left from the first backward pass are 9, 6, and -1. The maximum of these RS values is 12, which is the same as the node's starting RF value. Thus, the RF value of node (2,4) also does not change.

In a like manner, none of the other RF values for the nodes in the second column from the right change during the second backward pass. The RF value for node (3,4) remains at 12. The RF value for node (4,4) remains at 8. And the RF value for node (5,4) remains at 3.

Once all of the new RF values for the second column from the right have been determined, new RS values for each node in the column are calculated. Required Start equals Required Finish minus cost.

The third column from the right (column 3) is updated next. Following the same process as used in the previous column, neither node (1,3) nor node (2,3) have changes to their RF values. However, node (3,3) experiences a change. Previously, node (3,3) had an RF value of 3. From the just-updated column to its right, RS values of 3, 2, and 0 adjoin it. The nodes immediately above and below it have RS values of 6 and 0 from the previous backward pass. The adjacent nodes in the column to its left have RS values of 7, 4, and -4 from the previous backward pass. The maximum of these RS values is 7, so the RF value of node (3, 3) changes to 7. For clarity on the graphics for Figure 39, both the updated RF value and the RS values from which it came are highlighted by diamonds. Following the same methodology, the RF value of node (3,4) changes to 4, due to the RS value of node (3,3) above and to its left. The changed RF value and its source RS value are circled in Figure 39. Node (3, 5) does not have changes to its RF value. Finally, once all of the RF values for the third column from the right have been determined, new RS values are calculated for each node by subtracting cost from RF.

Next, updates begin on RF values for the fourth column from the right (column 2). For the second backward pass of this example, three nodes change their RF values. The RF value of node (3, 2) is increased to 7 due to the RS value of node (2, 2) above it.

This change is designated by squares on Figure 39. Likewise, the RF values of nodes (4, 2) and (5, 2) change to 4 and 2, respectively, with the changes highlighted by triangles and crosses. After all RF values in the column are updated, RS values are calculated for each node in this column.

The process continues methodically from right to left until all columns have been updated for the entire array. In this example, for the fifth and final column from the right, both nodes (4, 1) and (5, 1) change their RF values compared to the previous backward pass, increasing to 5 and 1, respectively, as shown in Figure 39. Likewise, only nodes (4, 1) and (5, 1) have new RS values.

If no changes are made to any RF or RS values in the entire matrix during the second backward pass, then the array is optimized. If changes were made, as they are in the above example within three of the columns, then an additional backward pass is needed to move towards an optimized route through the matrix.

7.3.4 Additional Backward Passes

The same methodology is used for the third backward pass through the matrix, with the results of the second backward pass providing the starting RF and RS values for each node. The RF value for each node in a column is updated based on the RS values of the 8 (or 5) nodes adjacent to it. Once all the nodes in a column are updated with new RF and RS values, the values for the next column to its left are determined. Updating of the array's RF and RS values proceeds from the right-most column to the left, with one column at a time updated.

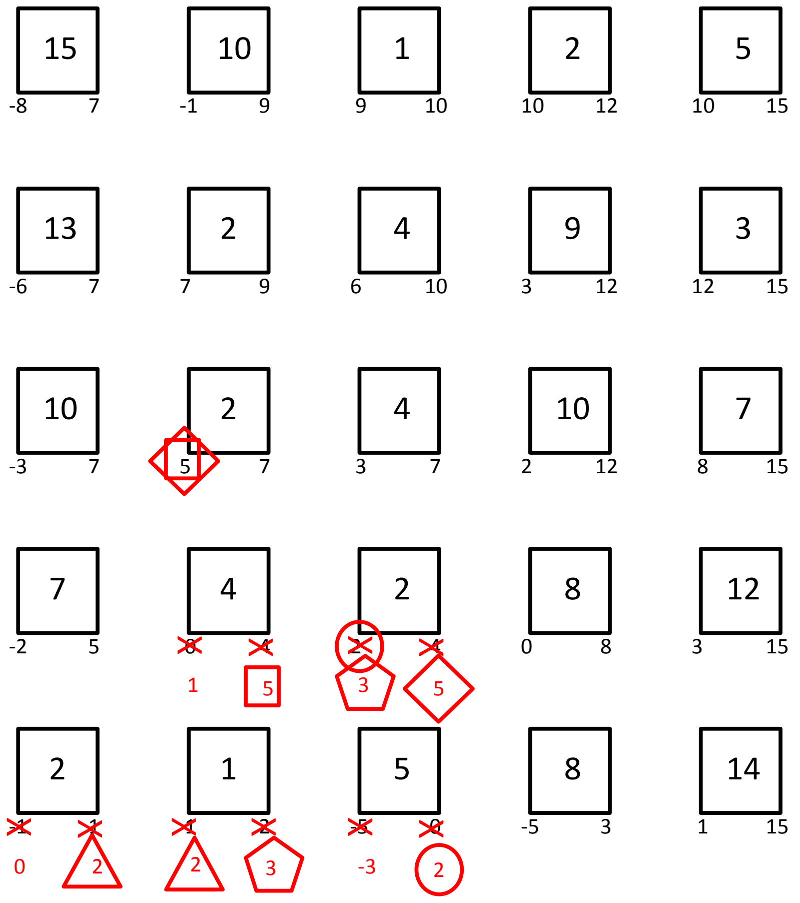


Figure 40: Third Backward Pass RS and RF Values for Example 5x5 Matrix

The third backward pass through the example 5x5 matrix gives the results shown in Figure 40. Five nodes had change in their RS and RF values compared to the previous pass, so another backward pass is needed to move closer to the solution.

The fourth backward pass through the example 5x5 matrix gives the results shown in Figure 41. Only one node had a change in its RS and RF values compared to the previous pass. However, another backward pass is needed to confirm whether or not the Optimized Transit Path has been determined.

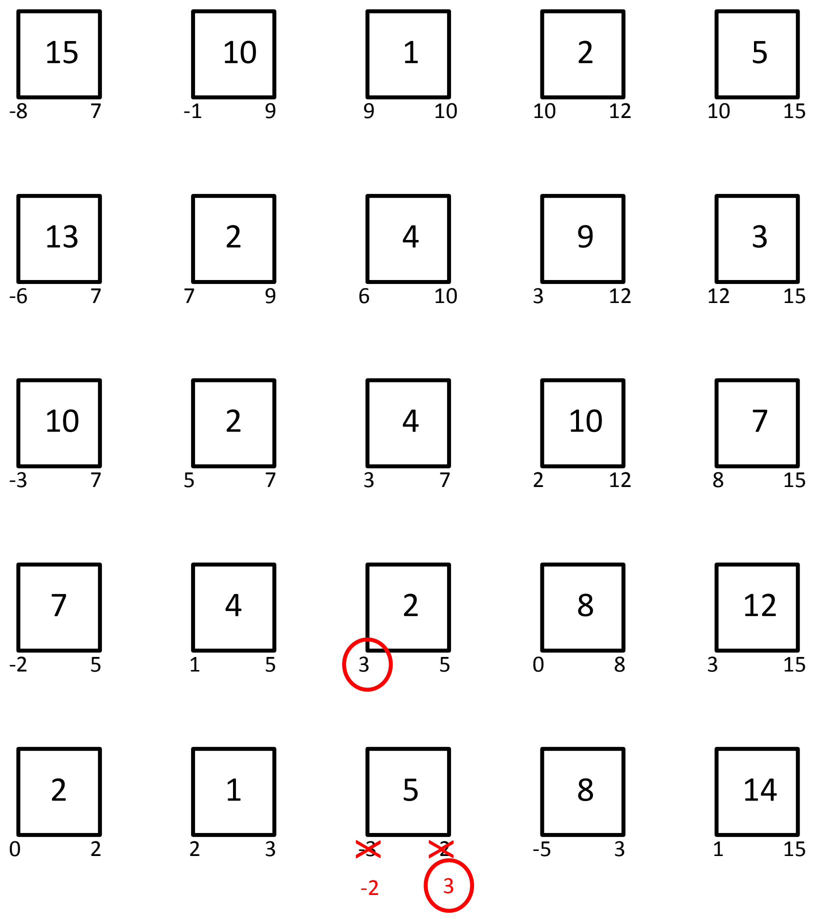


Figure 41: Fourth Backward Pass RS and RF Values for Example 5x5 Matrix

The results for the fifth backward pass (Figure 42) are identical to those for the fourth. This means that the RF and RS values for all nodes in the matrix have been optimized for their maximum possible values.

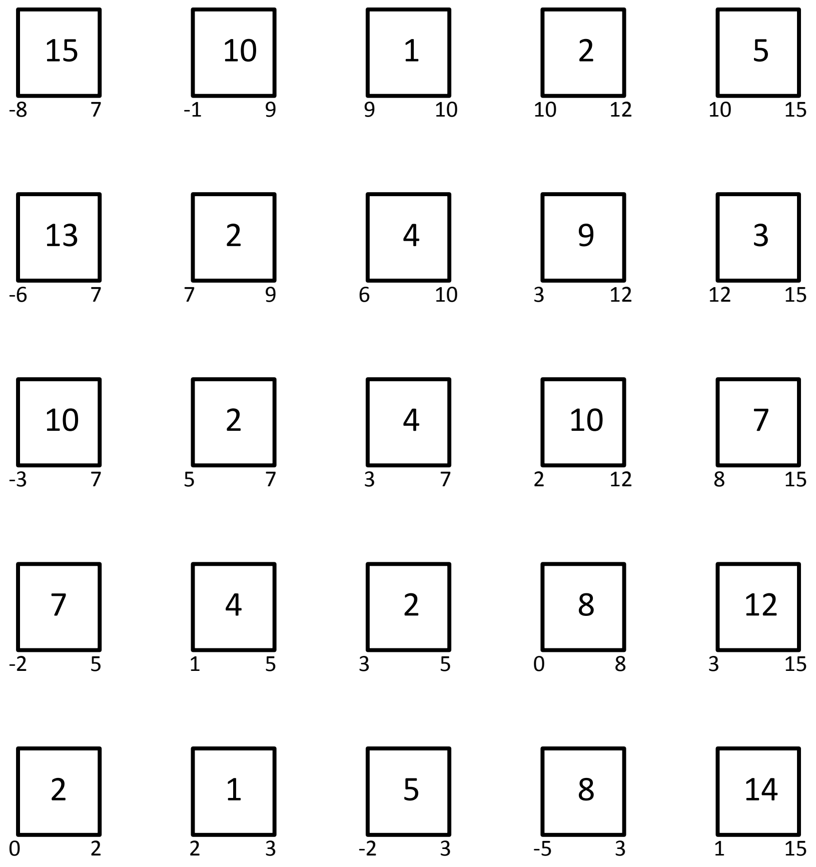


Figure 42: Fifth Backward Pass RS and RF Values for Example 5x5 Matrix

7.4 Identifying the Optimized Transit Path

7.4.1 Determining Optimized Slack

Once the optimized path backward through the array has been determined, Optimized Slack (OS) can be calculated for each node. Optimized Slack for a node equals the difference between its Required Finish and its Minimum Finish values.

$$OS = RF - MF \quad (23)$$

Alternatively, Optimized Slack for a node also equals the difference between its Required Start and its Minimum Start.

$$OS = RS - MS \quad (24)$$

Graphically, Optimized Slack is represented in Figure 43. The top row in the figure represents Forward Pass calculations for the shortest path through the matrix that

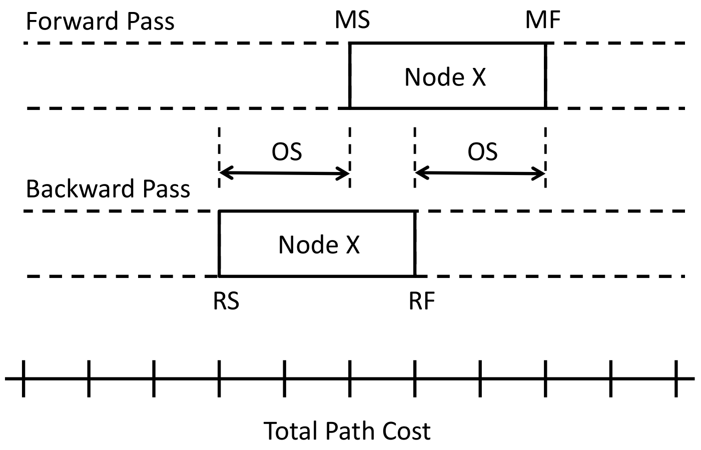


Figure 43: Graphical Representation of Optimized Slack

includes Node X. The bottom row represents Backward Pass calculations that include Node X. The difference in the location of Node X between the rows is the Optimized Slack for that node.

Optimized Slack (OS) is analogous to Total Slack (or Total Float) used in Critical Path Method (CPM) scheduling. In CPM, the Total Slack for a node represents the difference between the required completion time of the project and the maximum length path that passes through that node. In the OTP algorithm, the Optimized Slack for a node represents the difference between the shortest possible path through a matrix and the shortest path through the matrix that includes that node. Thus, OS indicates the minimum "penalty" associated with diverting to a particular node. The additional insight provided by Optimized Slack is one of the greatest strengths of the OTP algorithm compared to traditional shortest path methods. Classical methods do not quantify the additional cost of deviating from the shortest path.

For the example 5x5 matrix, Optimized Slack for each node, along with MS, MF, RS, and RF, is shown in Figure 44.

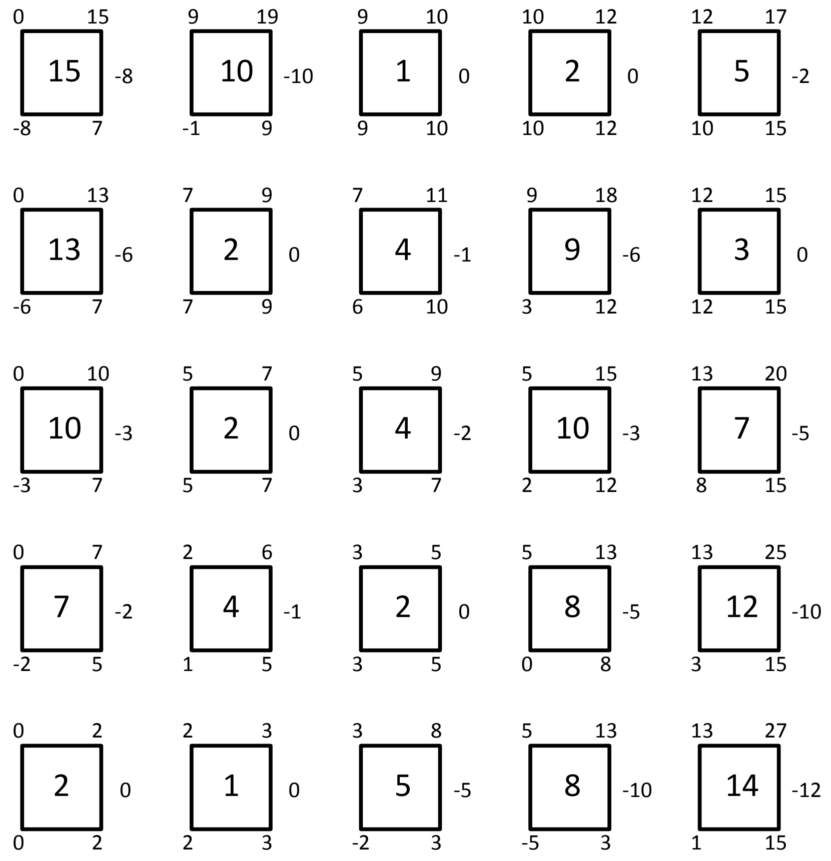


Figure 44: Optimized Transit Path MS, MF, RS, RF, and OS Values for Example 5x5 Matrix

7.4.2 Identifying the Optimized Transit Path

Nodes with Optimized Slack values of zero are on the Optimized Transit Path through a matrix. (This is equivalent to how nodes with minimum values of Total Slack are on the critical path in CPM scheduling.) A sequence of nodes, each with Optimized Slack of zero, should be followed in order to minimize the total cost in passing through the array. Figure 45 shows the Optimized Transit Path through the example 5x5 matrix.

For a particular node, a negative value of Optimized Slack represents the minimum additional cost to transit through the matrix if that node were included as a waypoint. Positive values of Optimized Slack do not occur.

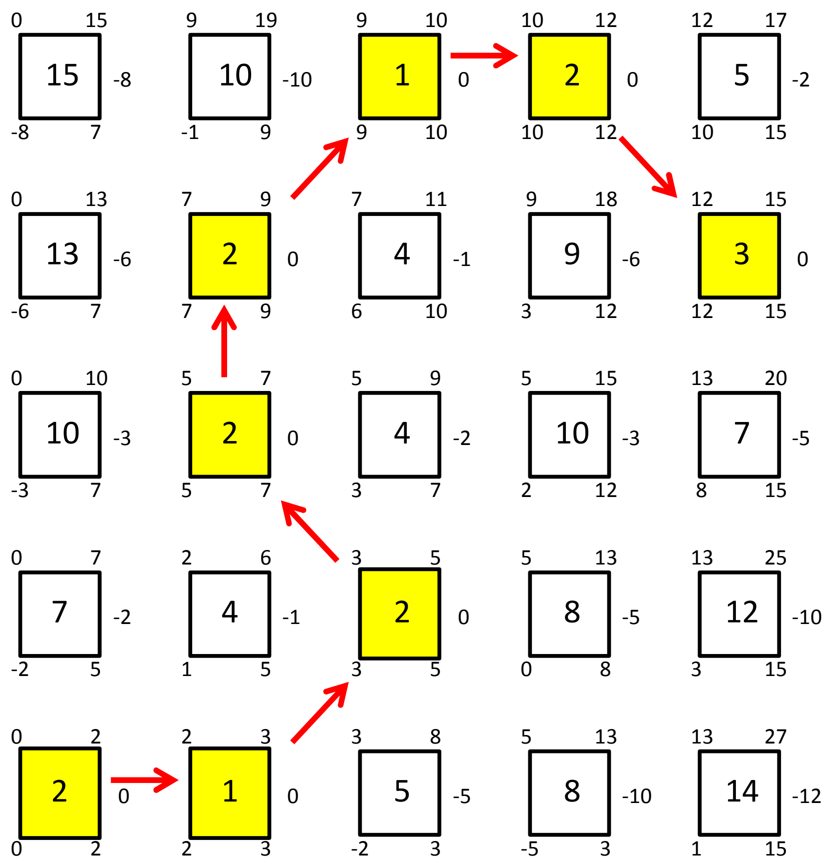


Figure 45: Optimized Transit Path through Example 5x5 Matrix

7.5 Variations to the Optimized Transit Path Algorithm

7.5.1 Defining a Specific Start and/or End Point

At times, it may be desired to start and/or end from a specific point when calculating the Optimized Transit Path through a matrix. For example, in the piracy application, a mariner already in transit might want to calculate the lowest cost path a starting point of his current position to the "finish line" on other side of the Arabian Sea, where he will have cleared the piracy danger zone. Likewise, a mariner preparing to enter the Arabian Sea might have the flexibility to enter from any point on the "starting line" but have a specific port that he must transit to as his ending point. In still other applications, a mariner could desire to know the Optimized Transit Path from his current starting position to a defined finish location, in which case both a desired start and end point would be specified.

A specific start and/or end point can be easily incorporated into shortest path calculations for the Optimized Transit Path algorithm via the following steps.

1. Add a column either ahead of the starting line (the first column) or after the finish line (the last column), or both, depending on whether a specific start and/or end point is desired. Set the cost value of every node in this column equal to infinity, except for the node that is horizontally adjacent to the desired start/end node. Set the cost value of this adjacent node equal to the cost of the desired start/end node.
2. Set the cost value of the desired start/end node equal to zero.
3. Solve the Optimized Transit Path algorithm for the entire matrix, with the extra column(s) included.

The shortest path will definitely include the adjacent node in the added column, as it is the only node in the column with a non-infinite cost. The path will also travel through the actual, desired start/end point because it has zero cost, so it does not add to the path length. The total, overall cost of the Optimized Transit Path will be correct because the sum of costs of the non-infinite node from the added column, plus the zero cost of the desired start/end point still equals the true cost of the desired start/end point.

For example, for the example 5x5 matrix previously analyzed in Sections 7.1 to 7.4, if it was desired to start the transit at node (2,1) (e.g. because that is a merchant ship's current position), the process would be as follows.

1. Add a column to the left of the matrix with infinity cost values for all nodes except the second row. Set this node's cost equal to that of the desired start point, which is 13.
2. Set the cost value of the desired start point equal to zero.
3. Solve the Optimized Transit Path algorithm for the new matrix for the MS, MF, RS, RF, and OS value of each node.
4. Identify the Optimized Transit Path, which consists of those nodes with OS values of zero. This path will pass through the desired start point.

Figure 46 shows the setup of the matrix with the additional column and the editing of the cost of the desired start point node. Figure 47 shows the MS, MF, RS, RF, and OS values for each node, with the Optimized Transit Path highlighted. The impact of specifying a specific starting point on computation time will be discussed in Section 9.3.1.

∞	15	10	1	2	5
13	0	2	4	9	3
∞	10	2	4	10	7
∞	7	4	2	8	12
∞	2	1	5	8	14

Figure 46: Editing of Example 5x5 Matrix to Include a Specific Start Point

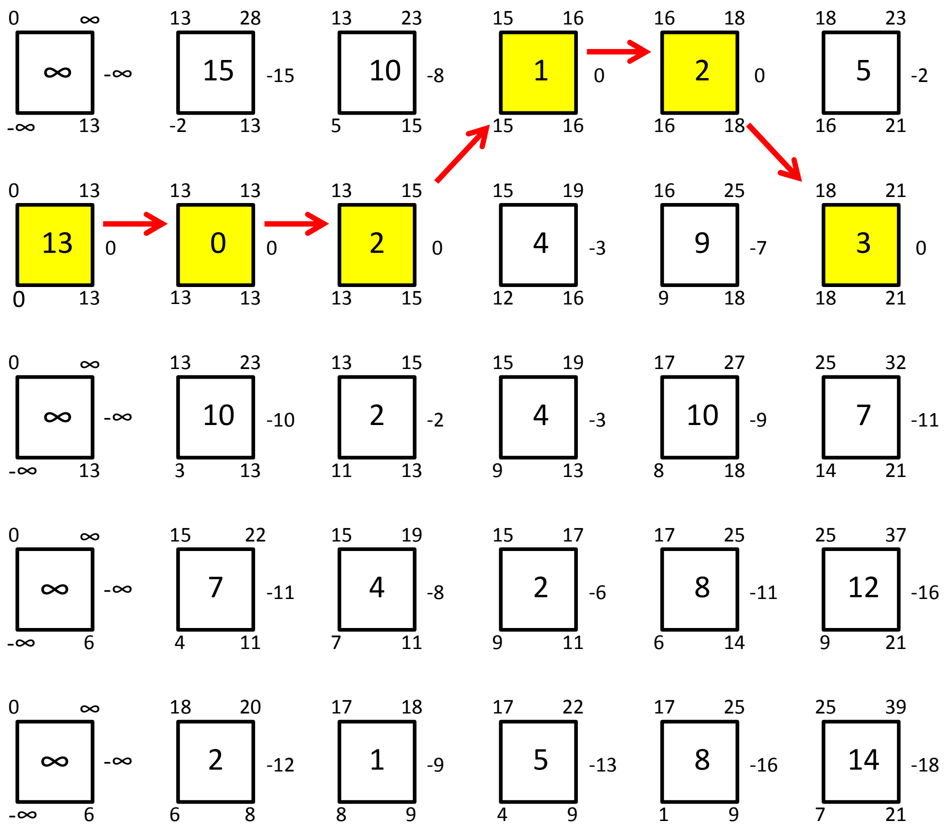


Figure 47: Optimized Transit Path Solution of Example 5x5 Matrix to Include a Specific Start Point

7.5.2 Limiting Directions of Motion

At times, it may be beneficial to limit motion through the matrix to only the horizontal (left/right) and vertical (up/down) directions. Thus, from each node, travel would be limited to only four potential directions, rather than eight. In the piracy application, an example of when this option might be desirable is when transit cost is included in consideration of the Optimized Transit Path. Because transit cost is directly related to distance traveled, in order to keep the transit cost consistent between any two adjacent nodes, the distance traveled between them must remain constant. Within the PARS matrix of piracy probability values, horizontally or vertically adjacent nodes are 50 miles apart from center to center. Meanwhile, diagonally adjacent nodes are $\sqrt{2}$ * 50 miles = 70.5 miles apart from center to center.

If transit costs are considered, an easy way to keep node-to-node distances equal is to limit travel to only the horizontal and vertical directions. The Optimized Transit Path algorithm easily incorporates such a limitation. In order to restrict travel to equal distances between nodes, on each forward and backward pass, nodes should look only to their immediate left and right, as well as directly up and down, for potential updates to their MS and RF values.

For the example 5x5 matrix previously analyzed, restricted motion through the matrix would be limited to only the directions shown in Figure 48. Note that the diagonal directions of motion have been removed from consideration. The Optimized Transit Path algorithm proceeds in the same manner as previously discussed. The only difference is that each node only looks at four adjacent nodes for updates, rather than eight.

Figure 49 shows the MS, MF, RS, RF, and OS values for each node in the example 5x5 matrix, as well as the Optimized Transit Path, when motion is restricted to the horizontal and vertical directions. Note that the overall cost of the Optimized Transit Path goes up slightly, since the path is no longer able to “cut the corner” by traveling diagonally between nodes. The impact of limiting motion direction on computation time will be discussed in Section 9.3.2.

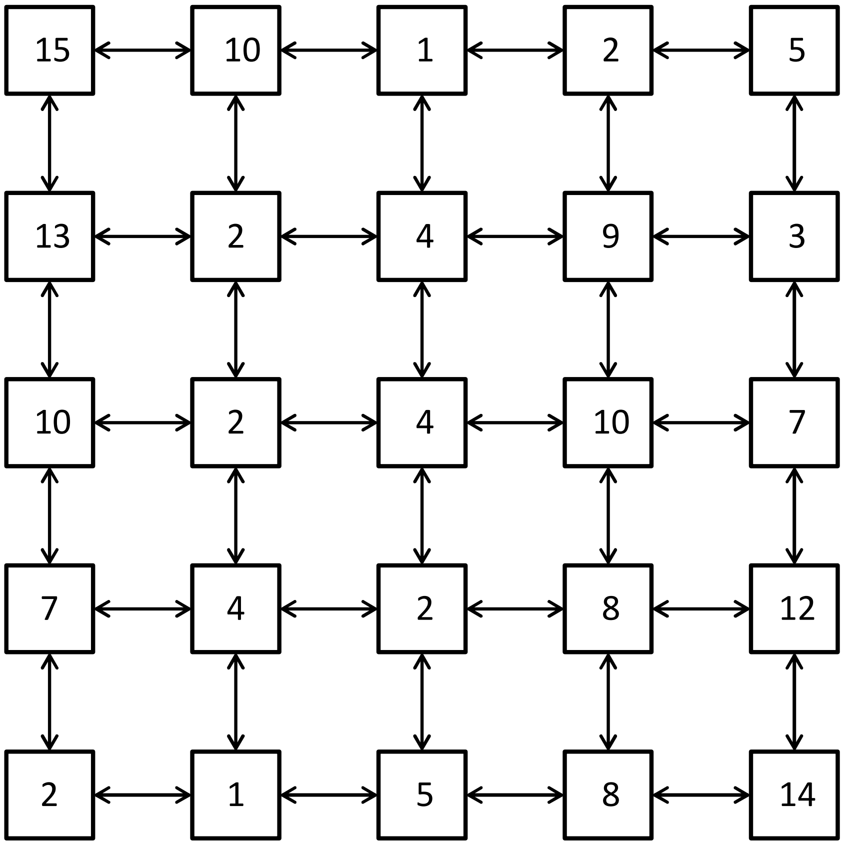


Figure 48: Limited Motion Paths Between Nodes in Example 5x5 Matrix

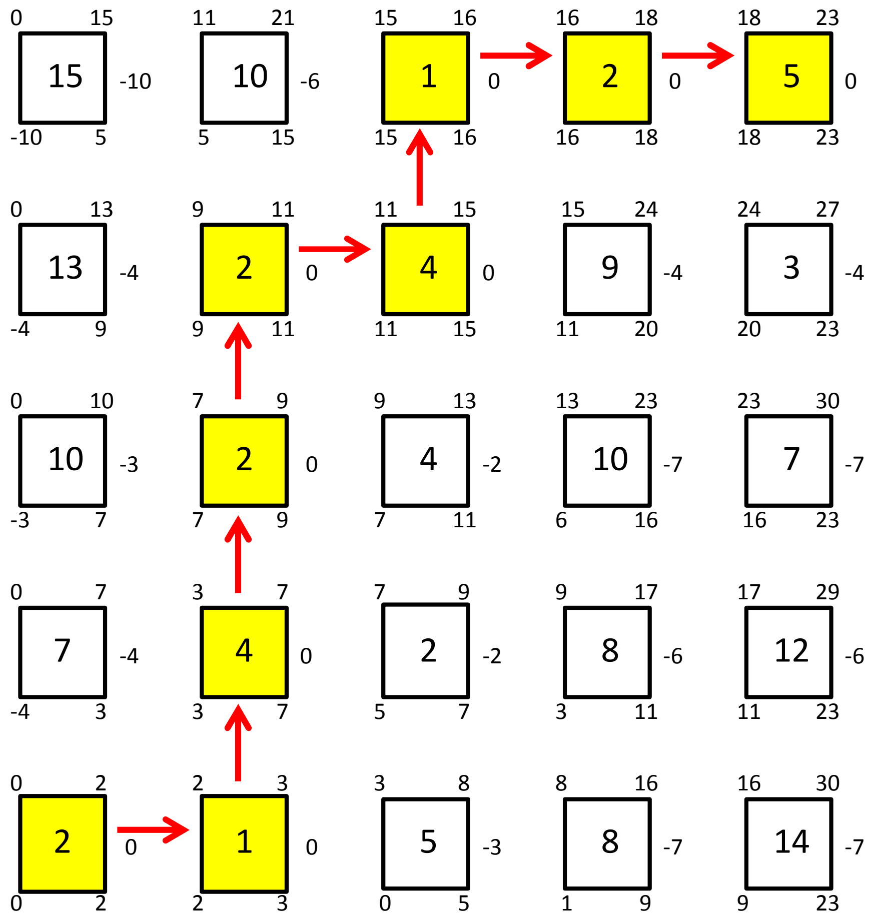


Figure 49: Optimized Transit Path Solution of Example 5x5 Matrix to Include Limited Motion

Chapter 8: Computer Modeling of the OTP Algorithm

In order to quantify the performance of the Optimized Transit Path algorithm, a computer model was written in MATLAB based on its logic. One of MATLAB's greatest strength is in matrix calculations, so it is well suited for the required calculations of this research. In addition, MATLAB is readily available and is able to run on computers with moderate computing power. One sub-goal in developing a computer algorithm for this research was to minimize computer processing requirements, so that relatively inexpensive processors could perform the needed calculations. Writing the code in MATLAB lends itself well to achieving this goal.

The complete MATLAB code written for this study, annotated with comments, is included in Appendix 1. The subroutines of this code are detailed in Appendices 1A through 1M. In generic terms, the computer code consists of the following logic.

A. INPUT AND PREPARE THE MATRIX TO BE ANALYZED

a. Input the external matrix that an Optimized Transit Path is desired for (Appendix 1A)

b. Start a timer so that the overall computation time can be calculated.

c. Create bounding rows along the top and bottom of the matrix with very high cost values in each node. By adding these extra rows, every row in the actual matrix can follow the same logic and look at 8 nodes surrounding it, rather than nodes in the top and bottom row of the matrix only looking at 5 nodes around them. Because the nodes in the bounding rows have very high values, the shortest path through the matrix will never want to include these nodes. (Appendix 1B)

B. FORWARD PASS CALCULATIONS

a. Set initial values of Minimum Start (MS) and Minimum Finish (MF) to be very high. These values will be lowered through successive forward passes. (Appendix 1C)

b. Begin a "for" loop that will repeat itself until the forward pass Optimized Transit Path solution is found.

    i. Copy the current values of MS and MF into two new matrices called Previous_MS and Previous_MF. These will be used to determine when to stop the "for" loop. When, at the end of a forward pass, the new value of MS or MF equals the previous value of MS or MF for every node in the matrix, then no further changes are possible, so the loop can stop. Until then, the "for" loop will continue repeating itself.

    ii. Conduct forward pass calculations, as discussed in Chapter 7, to determine new values of MS and MF for each node in the matrix. In these calculations, MS for each node is compared to MF of the nodes immediately surrounding it, and MS is lowered to match the minimum of the surrounding MF values. After new MS values are calculated for a column, its new MF values are calculated by adding node cost to MS. Columns are updated one at a time, from left to right through the matrix. (Appendix 1D)

iii. Compare new MS and MF values to Previous_MS and Previous_MF values. If they are identical for all nodes, stop the "for" loop. If they are not identical for all nodes, the "for" loop must be repeated again.

c. Count the number of forward passes and the time needed to arrive at the Optimized Transit Path solution values of MS and MF. (Appendix 1E)

C. BACKWARD PASS CALCULATIONS

a. Set initial values of Required Start (RS) and Required Finish (RF) to be very large negative numbers. These values will be increased through successive forward passes. Set values in the last column of the RF matrix equal to the Minimum Finish (MF) of the last node on the Optimized Transit Path, which equals the smallest MF value of nodes in the last column as found in the final forward pass. (Appendix 1F)

b. Begin a "for" loop that will repeat itself until the forward pass Optimized Transit Path solution is found.

i. Copy the current values of RS and RF into two new matrices called Previous_RS and Previous_RF. These will be used to determine when to stop the "for" loop. When, at the end of a backward pass, the new value of RS or RF equals the previous value of RS or RF for every node in the matrix, then no further changes are possible, so the loop can stop. Until then, the "for" loop will continue repeating itself.

ii. Conduct backward pass calculations, as discussed in Chapter 7, to determine new values of RS and RF for each node in the matrix. In these calculations, RF for each node is compared to RS of the nodes immediately surrounding it, and RF is increased to match the maximum of the surrounding RS values. After new RF values are calculated for a column, its new RS values are calculated by subtracting node cost from RS. Columns are updated one at a time, from right to left through the matrix. (Appendix 1G)

iii. Compare new RS and RF values to Previous_RS and Previous_RF values. If they are identical for all nodes, stop the "for" loop. If they are not identical for all nodes, the "for" loop must be repeated again.

c. Count the number of backward passes and the time needed to arrive at the Optimized Transit Path solution values of RS and RF. (Appendix 1H)

D. OPTIMIZED SLACK CALCULATIONS AND OPTIMIZED TRANSIT PATH RECOMMENDATIONS

a. Calculate Optimized Slack (OS) for every node in the matrix. Optimized Slack equals the difference between Required Finish (RF) and Minimum Finish (MF) for each node, or the difference between Required Start (RS) and Minimum Start (MS). Those nodes with OS values of zero are the nodes on the Optimized Transit Path through the matrix. Negative values of OS for all other nodes indicated the additional cost that would be incurred if that node were diverted to, in the most efficient manner possible. (Appendix 1I)

b. Strip the high value bounding rows from the MS, MF, RS, RF, and OS matrices. (Appendix 1J)
c. Measure the total time spent on all calculations since the start of the analysis. (Appendix 1K)
d. Present the final results to the user in a concise manner, which includes both the Optimized Transit Path as well as OS values for each node. (Appendix 1L)
e. If desired, write the OS values to an external source so that they may be used to best convey the Optimized Transit Path recommended route. (Appendix 1M)

At its most stripped-down level, the code for the Optimized Transit Path algorithm could be reduced to the following steps.

A. Input matrix from external source
B. Add bounding rows of high values to top and bottom of matrix.

C. Run forward pass calculations up to (Number of Rows) times, until new MS and MF values equal Previous_MS and Previous_MF values from the previous pass.

D. Strip off bounding rows from MS and MF matrices

This simplified code would give the lowest possible MS and MF value for each node. From the node on the "finish line," or far right column, with the lowest MF value, one could backtrack through the matrix to find the Optimized Transit Path by identifying the source of the MF value that determined each node's MS value. This is much the same way that the shortest path is found with Dijkstra's Algorithm and other shortest path methods.

Such a simplification would cut the number of required calculations in half, since it would eliminate backward pass calculations. However, this change would eliminate the ability to calculate Optimized Slack. OS is one of the most important features of the Optimized Transit Path algorithm. Not only does it clearly identify the shortest route through the matrix (those nodes with OS = 0), it also quantifies the penalty in additional cost associated with diverting from the shortest path. Such information is especially valuable to the piracy application. It gives a mariner quantifiable feedback as to the impact of his choices to divert from the recommended Optimized Transit Path. Thus, for the piracy application, it is recommended to run the entire model, including forward passes, backward passes, and Optimized Slack calculations.

Chapter 9: Performance Results of the OTP Algorithm

9.1 Structure of the Experimental Test Plan

The MATLAB computer code for the Optimized Transit Path algorithm, as described in Chapter 8, was tested with a series of progressively more complex matrices, for which shortest path solution was already known. These matrices were imported into MATLAB and solved via the computer code. The answers were compared to the known solutions, verifying that the code worked properly. These test matrices had shortest paths that required motion in all directions, including straight up and down as well as backtracking. The code worked properly to incorporate potential motion in all directions in its solution for the shortest path through the matrix.

All performance tests of the OTP algorithm were run using MATLAB on a laptop computer of moderate computing power. System characteristics of the test bed laptop computer were as follows.

Manufacturer:	Toshiba
Model:	Satellite P745
Operating System:	Windows 7 Home Premium
Processor:	Intel Core i3-2350M CPU @ 2.30 GHz
Installed Memory (RAM):	6.00 GB

To measure the performance of the OTP algorithm, one hundred randomly generated matrices of each of the following sizes were generated.

- To test the effect of overall matrix size on performance:
  - 10x10, 20x20, 30x30, 40x40, 50x50, 60x60, 70x70, 80x80, 90x90, 100x100
- To test the effect of number of rows on performance:
  - 10x50, 20x50, 30x50, 40x50, 50x50, 60x50, 70x50, 80x50, 90x50, 100x50
- To test the effect of number of columns on performance:
  - 50x10, 50x20, 50x30, 50x40, 50x50, 50x60, 50x70, 50x80, 50x90, 50x100

Thus, performance characteristics were recorded for 2,800 different, randomly generated matrices. The same 50x50 matrices were used in all three cases for consistency and comparison between data sets.

The Optimized Transit Path solution was determined for each of these 2,800 matrices. For each solution, the following data was recorded.

- Number of forward passes required to reach solution
- Time (sec) for forward passes to reach solution
- Number of backward passes required to reach solution
- Time (sec) for backward passes to reach solution
- Time (sec) overall to reach entire solution, including MS, MF, RS, RF, and OS

In addition, based on the number of forward and backward passes required for each matrix, the total number of calculations required to reach the solution was also determined. The methodology for determining number of calculations will be discussed in Section 8.3.

Finally, one thousand randomly generated 43x50 matrices were solved using the OTP algorithm, with the number of calculations required to reach the solution recorded for each matrix. This data was compared to the number of calculations that would be required to solve a 43x50 matrix with each of the commonly used shortest path algorithms discussed previously in Chapter 6.

9.2 Number of Forward and Backward Passes Required to Reach a Solution

During the Optimized Transit Path solution to each of the test matrices, both the number of forward passes and the number of backward passes required to reach the solution were recorded.

The number of forward passes required to reach a solution to the Optimized Transit Path algorithm was recorded for each of the 2,800 test matrices, with the results as shown in Figure 50. At every node count value, the number of forward passes varied greatly, suggesting that some of the randomly generated matrices were more complicated than others and required a greater amount of vertical motion and backtracking than others. At many of the node count values, there were two distinct datasets of test values recorded. This is because of the symmetrical nature of the test

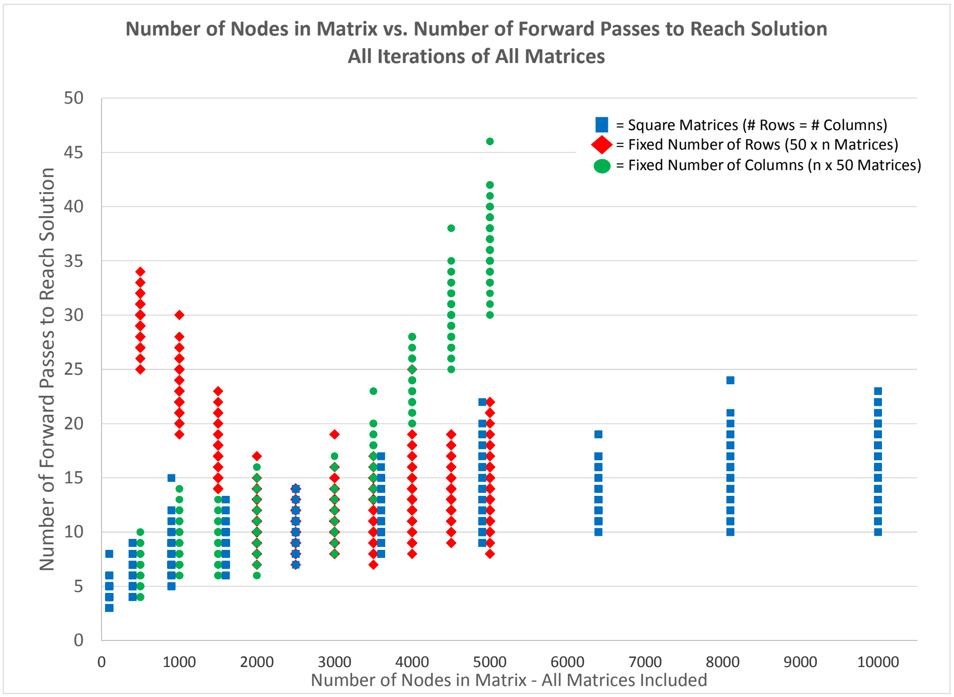


Figure 50: Number of Nodes vs. Number of Forward Passes – All Test Matrices

plan. For example, there is a set of 10x50 matrices and a set of 50x10 matrices, both of which have the same number of nodes, 500, in each matrix. At each of the node count numbers in which two different matrix sizes are represented, the results vary widely and suggest that there are two distinct sets of results for that number of nodes. This suggests that something beyond number of nodes influences the number of forward passes required to reach a solution.

To further investigate the effect of number of nodes on required number of forward passes, the test results for square matrices, those with an equal number of rows and columns, were plotted next. These matrices behaved in a much more linear fashion, as shown in Figure 51. This implies that, as the number or rows and the number of

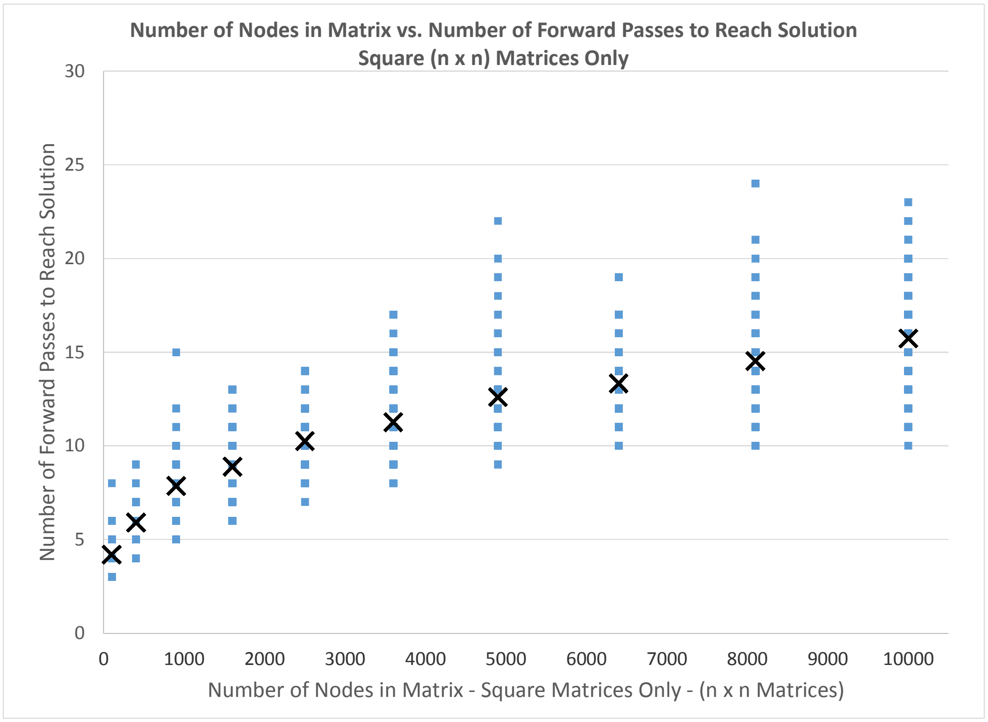


Figure 51: Number of Nodes vs. Number of Forward Passes – Square Matrices
(X = Mean Value for Each Number of Nodes)

columns increases, the required number of forward passes also increases. However, the results of Figure 50 suggest that one of these may have a greater impact on the results than the other.

In order to investigate the effect of number of rows on the required number of forward passes, results from those matrices with a fixed number of fifty columns and increasing numbers of rows were plotted, as shown in Figure 52. Likewise, results from matrices with a fixed number of fifty rows and increasing numbers of columns were also plotted, as shown in Figure 53.

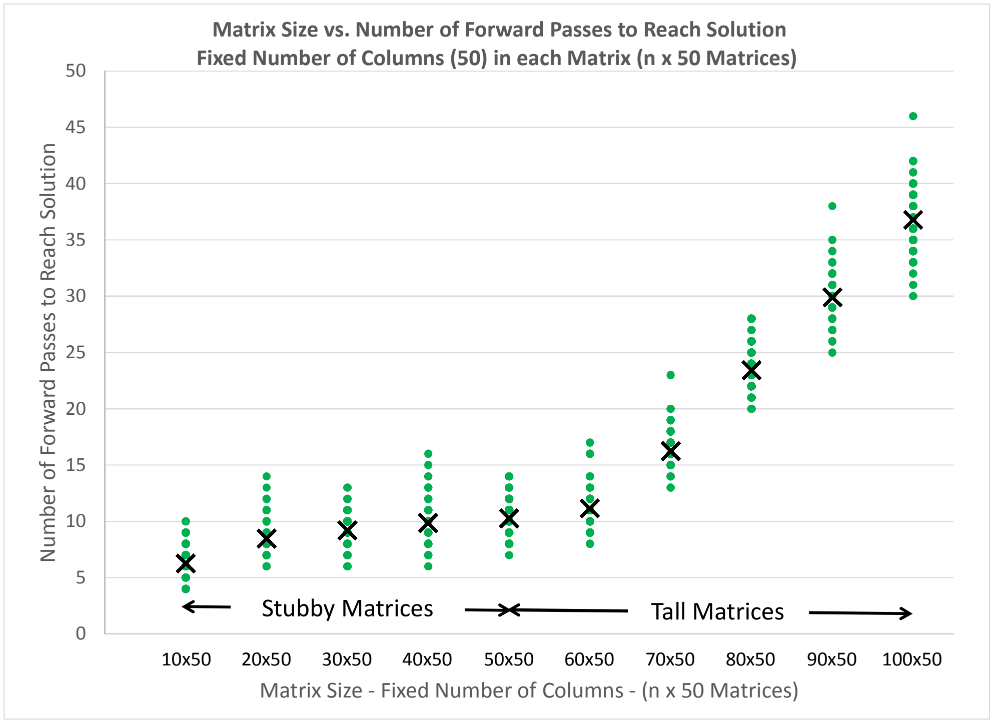


Figure 52: Matrix Size vs. Number of Forward Passes – Fixed # of Columns
(X = Mean Value for Each Matrix Size)

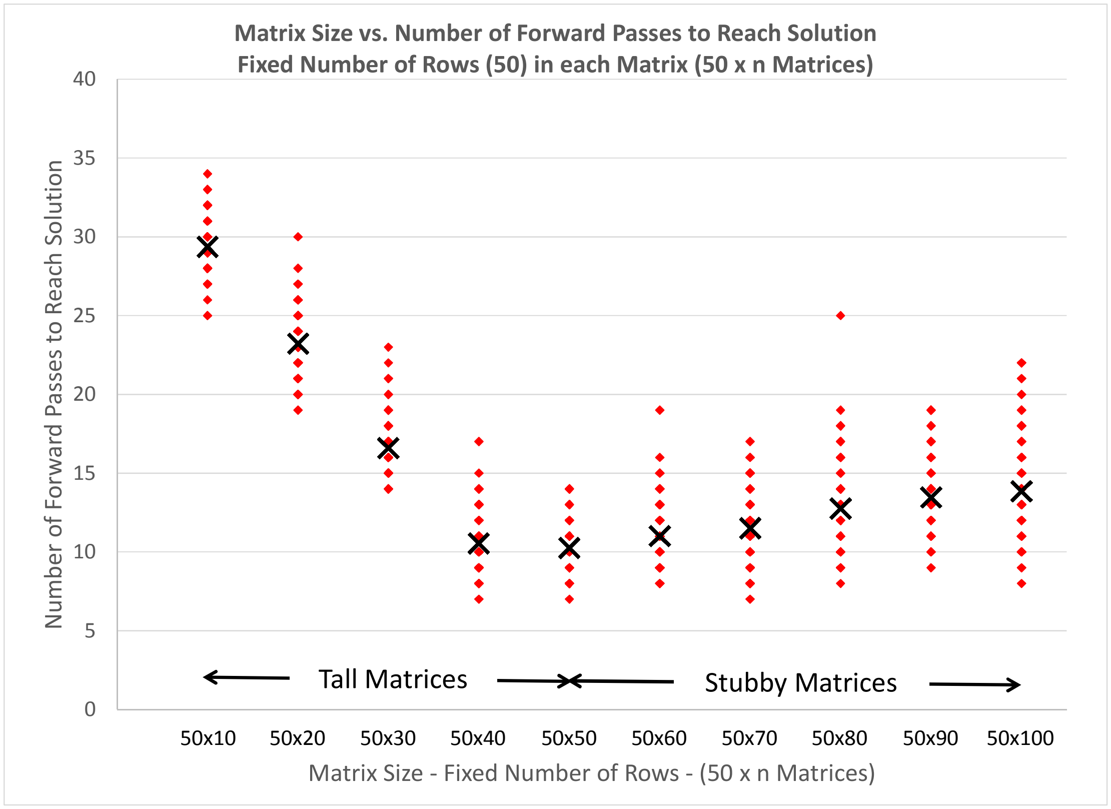


Figure 53: Matrix Size vs. Number of Forward Passes – Fixed # of Rows
(X = Mean Value for Each Matrix Size)

In Figure 52, in which the number of columns is held fixed while the number of rows is increased, the number of forward passes seems to increase quadratically (related to number of rows squared), with the required number of forward passes never exceeding the number of rows. Meanwhile, in Figure 53, in which the number of rows is held fixed while the number of columns is increased, the number of forward passes is quite high for the relatively short and stubby matrices (e.g. 50x10, 50x20, and 50x30), gradually decreasing as the matrix lengthens. Then, as the matrix approaches a square shape and lengthens past that, the number of required forward passes remains fairly constant, with a slight rise in both mean and standard deviation.

Typically, the number of forward or backward passes possible in an OTP algorithm solution will be less than or equal the number of rows in the matrix. This is because information within a particular column can only be transmitted straight up or down a distance of one node at a time within that particular column. Diagonal motion to subsequent columns speeds the exchange of information and reduces the required number of forward passes. However, if the shortest path involves significant vertical motion within a relatively small number of columns, this exchange of information will require a larger number of forward passes.

An example where the required number of forward passes would equal the number of rows in the matrix would be for an extremely short and stubby matrix, in which the entry point was at the absolute top of the first column and the exit point was at the absolute bottom of the last column. For example, consider a 50x3 matrix in which the first column consisted of all infinity valued nodes except the node in row 1, the last column consisted of all infinity valued nodes except the node in row 50, and the middle column consisted of all non-infinity nodes. Information about the first column's lone non-infinity node would require 49 forward passes to be transmitted to the last column's lone non-infinity node. Then, on the 50th forward pass, results would be identical to the 49th, so the algorithm would stop. Thus, for this extreme example, the number of required forward passes would equal the number of rows in the matrix.

However, for most matrices, the number of required passes will be much lower than the number of rows in the matrix. The experimental results from the tests of 2,800 matrices support this conclusion. Only in the most out-of-square matrices (e.g. 50x10, 50x20, 10x50, 20x50) did the required number of forward passes ever approach the number of rows. And only once of the 2,800 tests of randomly generated matrices (one instance in the hundred results for 10x50 matrices) did the number of forward passes ever equal the number of rows.

Experimental results for the number of backward passes required to reach a solution, as shown in Figure 54 through Figure 57, follow similar trends as the number of forward passes, though the effect of number of rows is not as pronounced. As with the forward passes, the number of backward passes needed to reach the OTP solution will typically be less than or equal to the number of rows. For the 2,800 test matrices, the number of backward passes never equaled the number of rows.

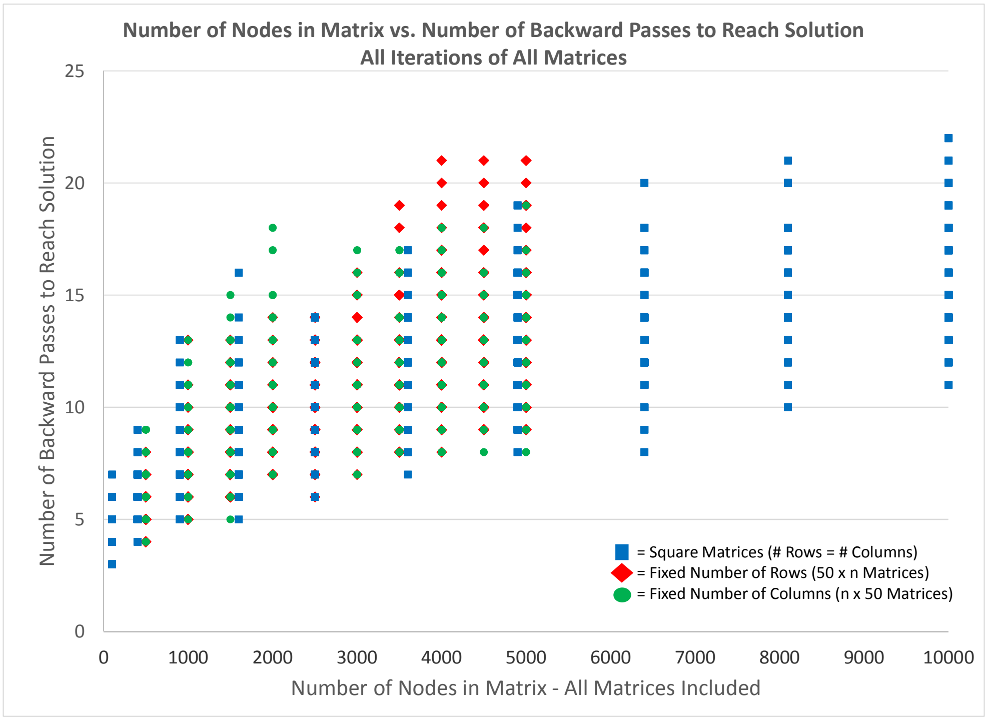


Figure 54: Number of Nodes vs. Number of Backward Passes – All Test Matrices

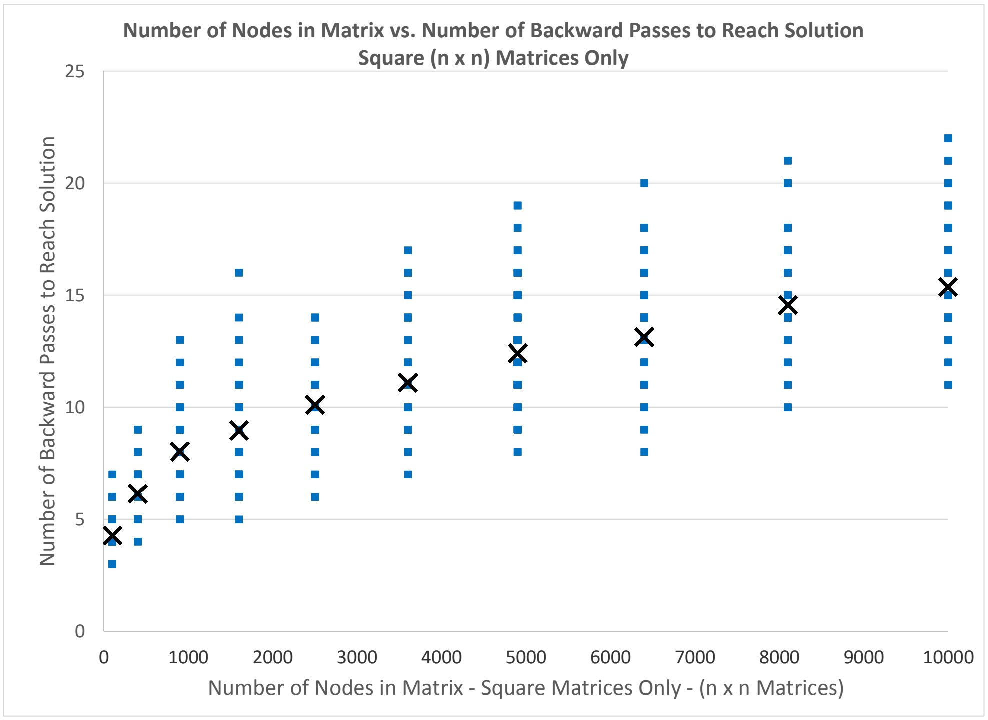


Figure 55: Number of Nodes vs. Number of Backward Passes – Square Matrices
(X = Mean Value for Each Number of Nodes)

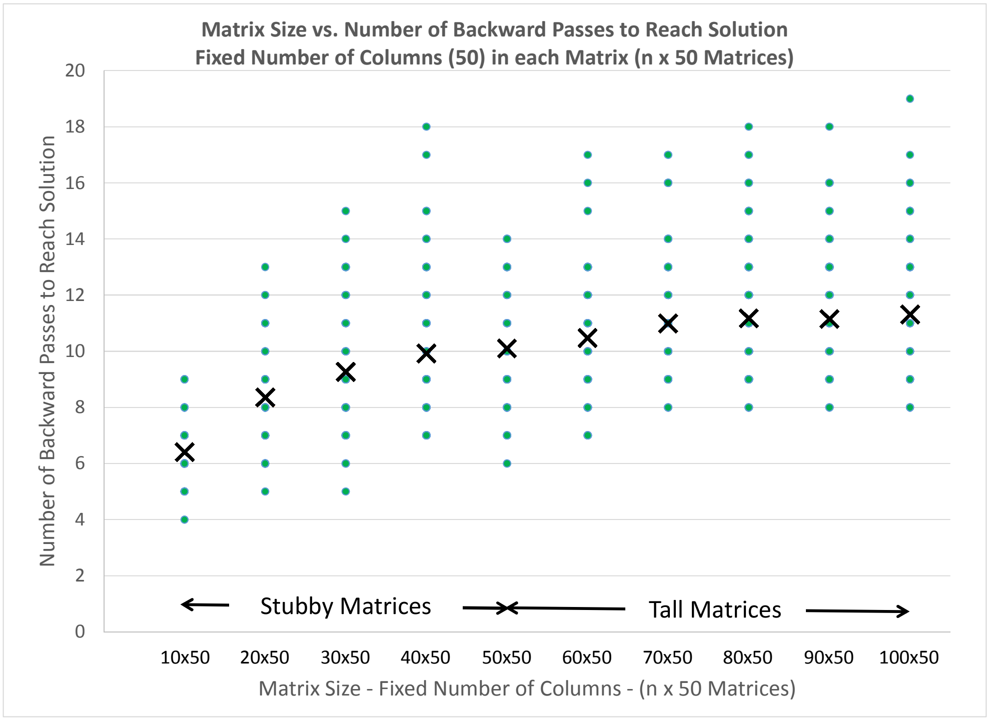


Figure 56: Matrix Size vs. Number of Backward Passes – Fixed # of Columns
(X = Mean Value for Each Matrix Size)

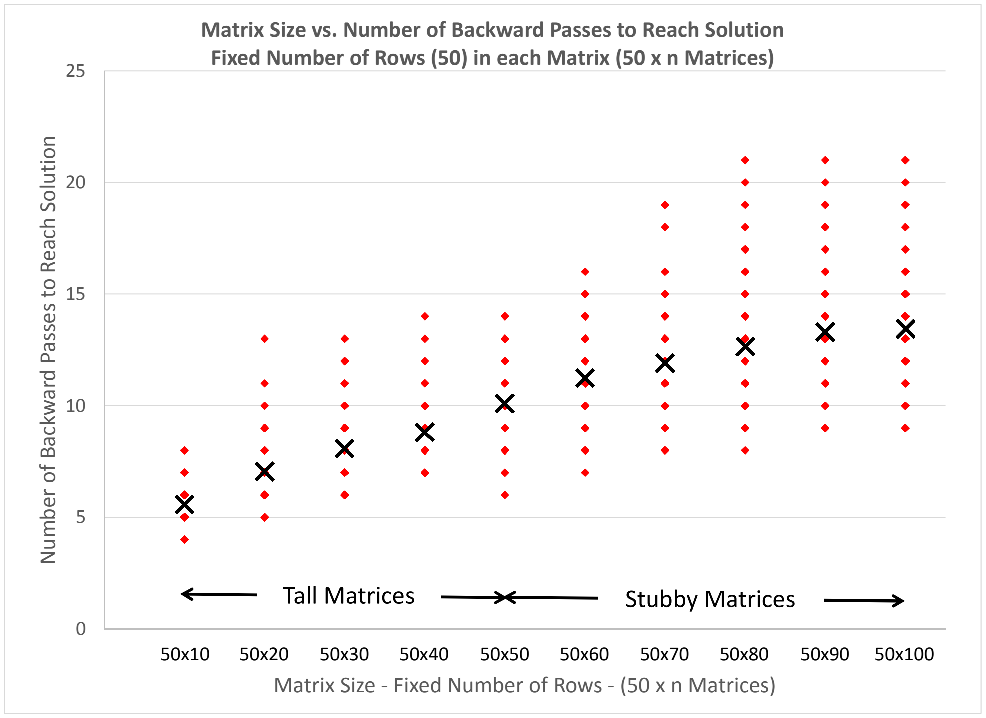


Figure 57: Matrix Size vs. Number of Backward Passes – Fixed # of Rows
(X = Mean Value for Each Matrix Size)

9.3 Number of Calculations Required to Reach a Solution

For each forward pass, there are nine calculations for each node in the "middle" columns of the matrix (8 calculations to check its MS value vs. MF values of the 8 nodes surrounding it and update its MS value as necessary; plus 1 calculation to determine its new MF value as MF = MS + Cost). There are six calculations for each node in the last column (5 to check its MS value vs. MF values of the 5 surrounding nodes and update it as necessary; plus 1 to calculate its new MF value as MF = MS + Cost). After the first pass, no calculations are needed for the first column, as its MS and MF values never change. However, in the computer code used to find the solution to the OTP algorithm, these values were recalculated as part of each pass, so there is one calculation for each node in the first row for each forward pass.

For R = Number of Rows, C = Number of Columns, and N = Number of Nodes, the number of calculations required to reach the Optimized Transit Path solution via forward passes is as follows.

$$\#Calc_{FwdPasses} = \#FwdPasses * [\#Calc_{first\ column} + \#Calc_{middle\ columns} + \#Calc_{last\ column}] \quad (25)$$

$$\#Calc_{FwdPasses} = \#FwdPasses * [R + (C-2) * R * 9 + 1 * R * 6]$$

$$\#Calc_{FwdPasses} = \#FwdPasses * R * [1 + (9C - 18) + 6]$$

$$\#Calc_{FwdPasses} = \#FwdPasses * R * [9C - 11] \quad (26)$$

The number of backward pass calculations required to reach the Optimized Transit Path solution values for RS and RF follows the same pattern as for the forward pass.

$$\#Calc_{BackPasses} = \#BackPasses * R * [9C - 11] \tag{27}$$

The overall number of calculations needed to reach the full Optimized Transit Path solution, including MS, MF, RS, RF, and OS values, equals the sum of the number of calculations for all forward passes, all backward passes, and Optimized Slack calculations. Since they are not done until all MS, MF, RS, and RF values are found for every node, finding OS only requires one calculation per node, either OS = RF – MF (Equation 23) or OS = RS – MS (Equation 24).

$$\#Calc_{Total} = \#Calc_{FwdPasses} + \#Calc_{BackPasses} + \#Calc_{OS} \tag{28}$$

$$\#Calc_{Total} = \#FwdPasses*R*[9C-11] + \#BackPasses*R*[9C- 11] + R*C \tag{29}$$

As expected, Figure 58 to Figure 61 show that the number of calculations required to find the Optimized Transit Path solution for each of the test matrices follow the same trends as the number of forward pass and number of backward pass figures.

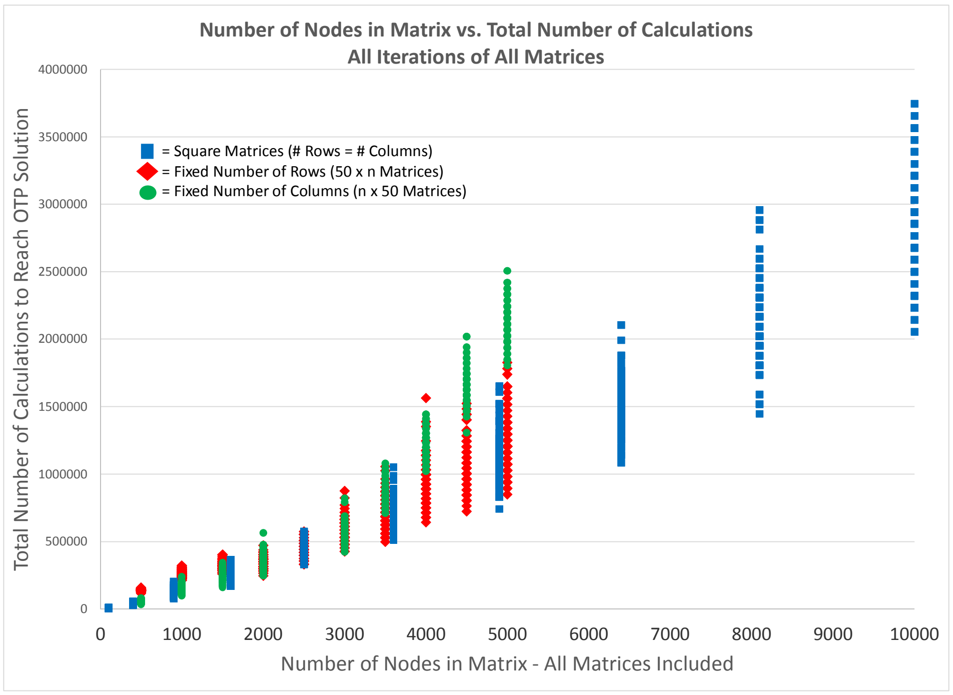


Figure 58: Number of Nodes vs. Number of Calculations – All Test Matrices
(Number of Calculations Determined per Equations 25-29)

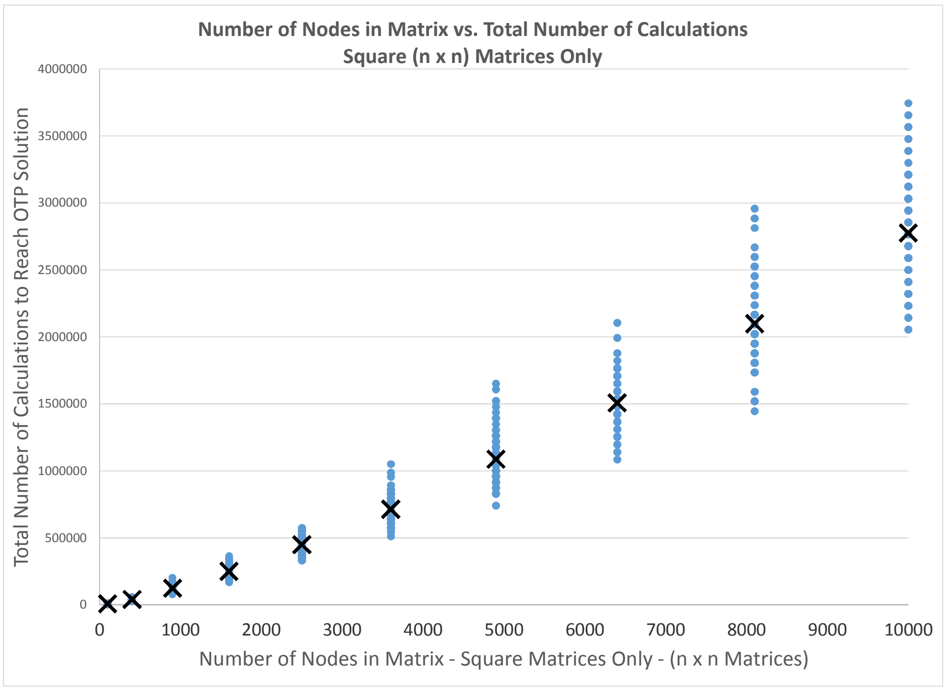


Figure 59: Number of Nodes vs. Number of Calculations – Square Matrices
(Number of Calculations Determined per Equations 25-29)
(X = Mean Value for Each Number of Nodes)

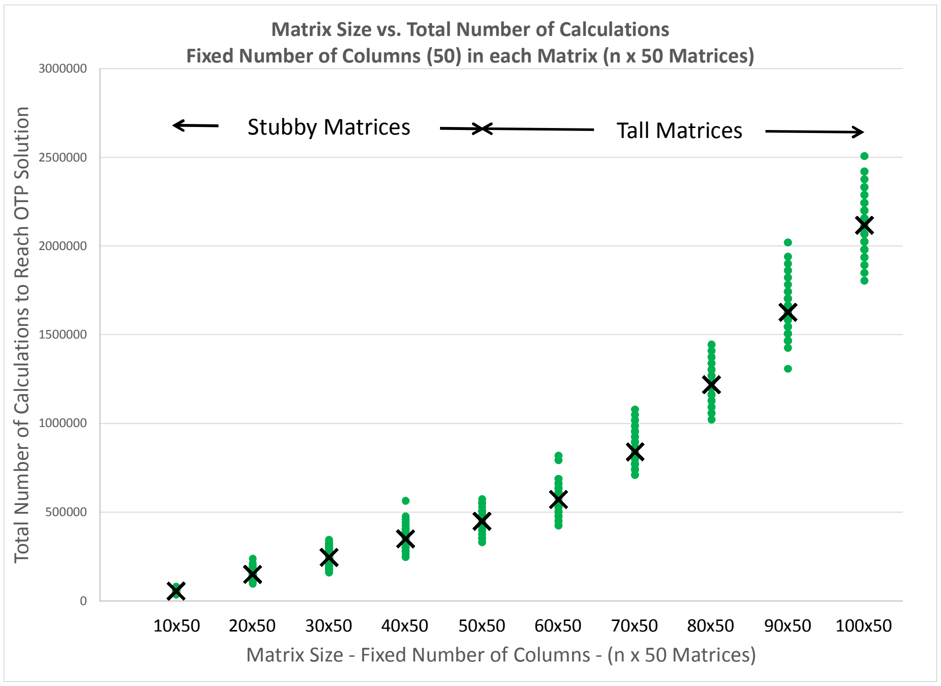


Figure 60: Matrix Size vs. Number of Calculations – Fixed # of Columns
(Number of Calculations Determined per Equations 25-29)
(X = Mean Value for Each Matrix Size)

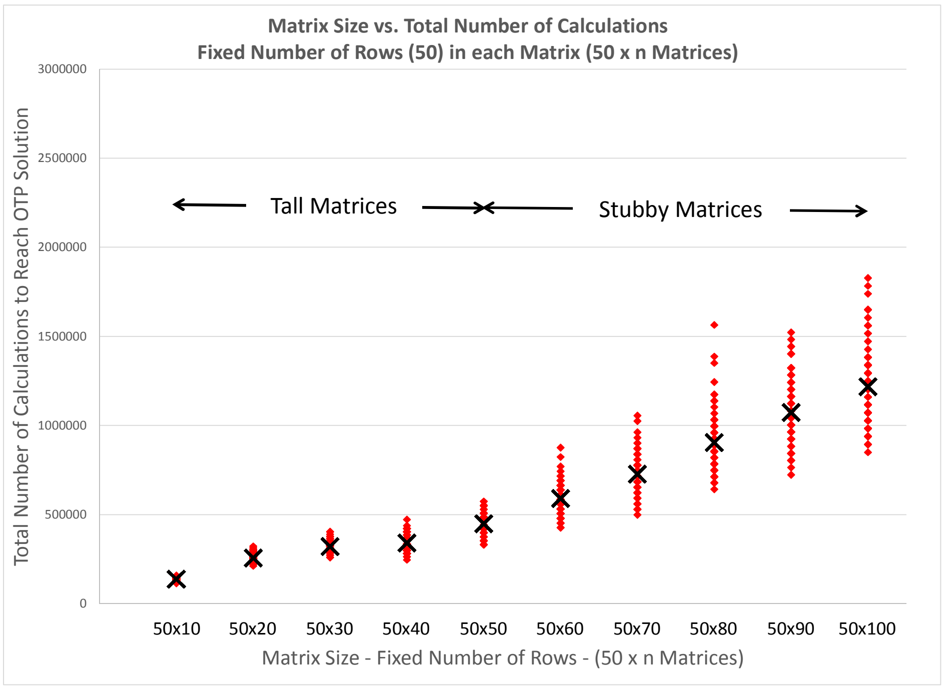


Figure 61: Matrix Size vs. Number of Calculations – Fixed # of Rows
(Number of Calculations Determined per Equations 25-29)
(X = Mean Value for Each Matrix Size)

9.3.1 Number of Computations to Solve the Specified Starting Point Case

Solving the Specified Point to Finish Line problem with the Optimized Transit Path algorithm, as previously discussed in Chapter 7.5.1, would have a minimal effect on the required number of calculations to reach a solution. Modifying the matrix to incorporate this requirement effectively adds one additional column to the matrix. Thus, (C + 1) is used instead of C within the equations to determine required number of calculations, as follows.

$$\begin{aligned}\#Calc_{Specific\text{-}Start\text{-}Point} &= \#FwdPasses*R*[\,9(C+1)-11] \\ &\quad + \#BackPasses*R*[\,9(C+1)-11] + R*(C+1)\end{aligned} \tag{30}$$

$$\begin{aligned}\#Calc_{Specific\text{-}Start\text{-}Point} &= \#FwdPasses * R * [\,9C-2] \\ &\quad + \#BackPasses * R * [\,9C-2] + R * (C+1)\end{aligned} \tag{31}$$

9.3.2 Number of Computations to Solve the Limited Motion Case

The other special case of the Optimized Transit Path algorithm, discussed in Section 7.5.2, was the limited motion problem, in which motion between nodes was restricted to only the 4 directions of up/down and right/left. In this case, the number of motion directions between nodes was reduced from 8 to 4 (or 5 to 3 for the first and last columns). Likewise, the number of calculations for each node during a pass through the matrix is reduced from 9 to 5 (or 6 to 4 for the last column). This change reduces the number of calculations required to reach the shortest path solution as follows.

$$\#\text{Calc}_{\text{FwdPasses-Limited Motion}} = \#\text{FwdPasses} * [R + (C\text{-}2) * R * 5 + 1 * R * 4] \quad (32)$$

$$\#\text{Calc}_{\text{FwdPasses-Limited Motion}} = \#\text{FwdPasses} * R * [5C - 6] \quad (33)$$

$$\#\text{Calc}_{\text{BackPasses-Limited Motion}} = \#\text{BackPasses} * [R + (C\text{-}2) * R * 5 + 1 * R * 4] \quad (34)$$

$$\#\text{Calc}_{\text{BackPasses-Limited Motion}} = \#\text{BackPasses} * R * [5C - 6] \quad (35)$$

$$\#\text{Calc}_{\text{Total-Limited Motion}} = \#\text{Calc}_{\text{FwdPasses}} + \#\text{Calc}_{\text{BackPasses}} + \#\text{Calc}_{\text{OS}} \quad (36)$$

$$\#\text{Calc}_{\text{Total-Limited Motion}} = \#\text{FwdPasses}*R*[5C–6] + \#\text{BackPasses}*R*[5C–6] + R*C \quad (37)$$

9.4 Computing Time Required to Reach a Solution

During the Optimized Transit Path solution to each of the 2,800 test matrices, a timer was started as soon as the matrix was generated. The time was recorded for all forward pass calculations, all backward pass calculations, and the overall solution time. For all three cases, the relationship between matrix size and solution time follows the same patterns as the relationship between matrix size and number of calculations. This is to be expected, as the dominant activities that require computing time are the matrix calculations. In Figure 62 to Figure 65, the effect of number of nodes and matrix size on total solution time is presented.

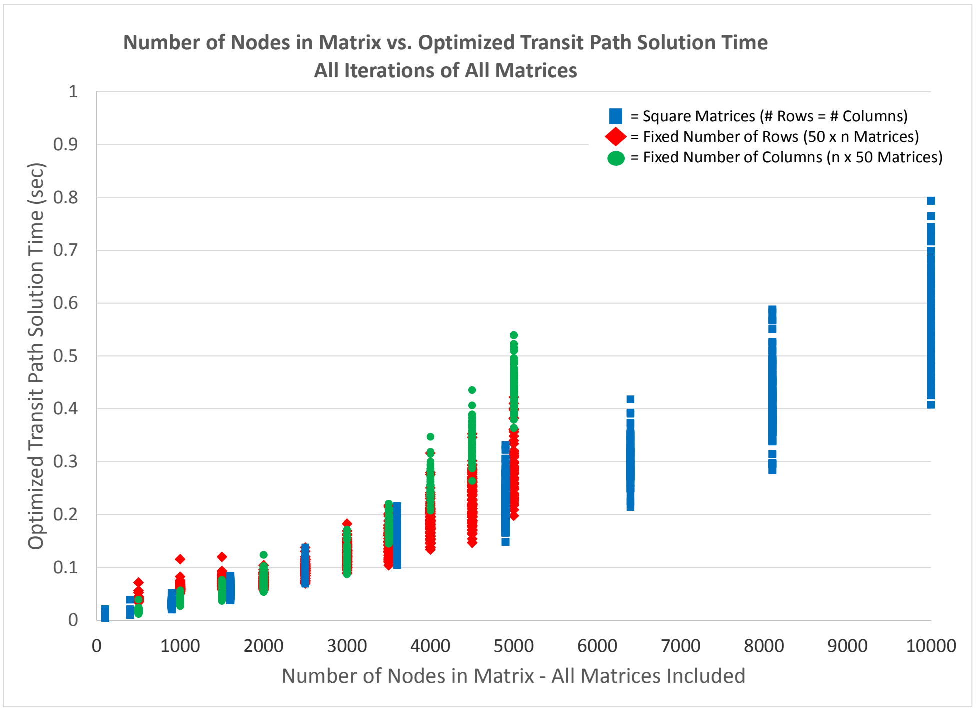


Figure 62: Number of Nodes vs. Calculation Time – All Test Matrices

It is worth noting that the computing time for the OTP Algorithm is extremely quick. Though a somewhat outdated laptop was running the code, the solution times for all matrices were but fractions of a second. Even the largest of the matrices, the 100x100 matrix with 10,000 nodes, averaged just over half a second for its total solution time, with a maximum value of 0.8 seconds. Thus, for the size of matrices involved in the piracy avoidance problem, total solution time for the OTP algorithm is not likely to be an issue for even older computers found aboard merchant ships.

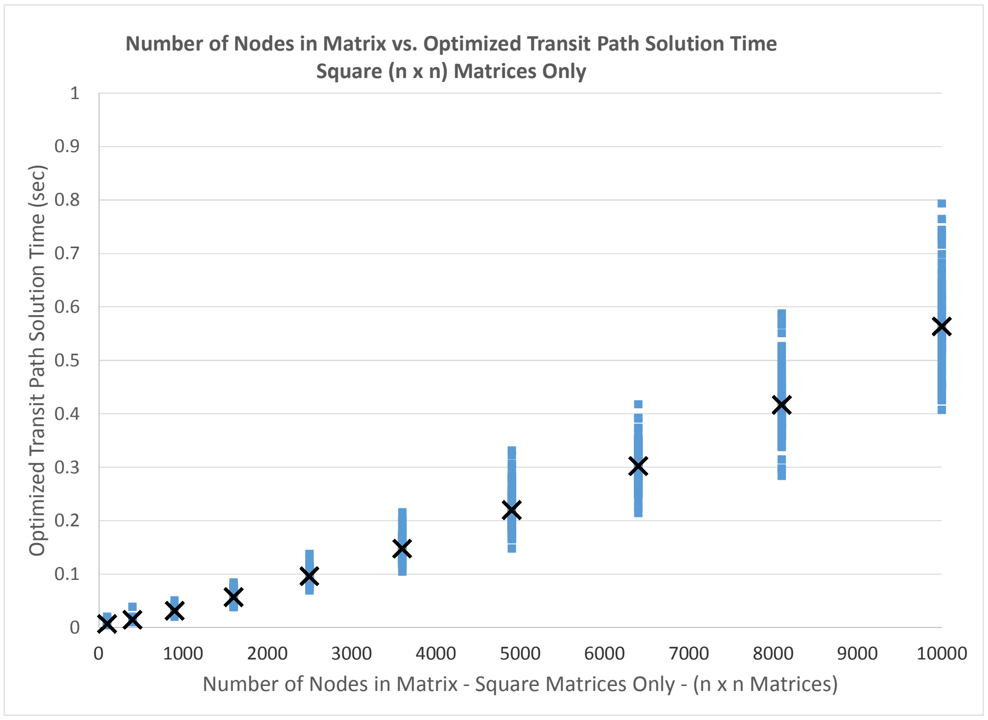


Figure 63: Number of Nodes vs. Calculation Time – Square Matrices
(X = Mean Value for Each Number of Nodes)

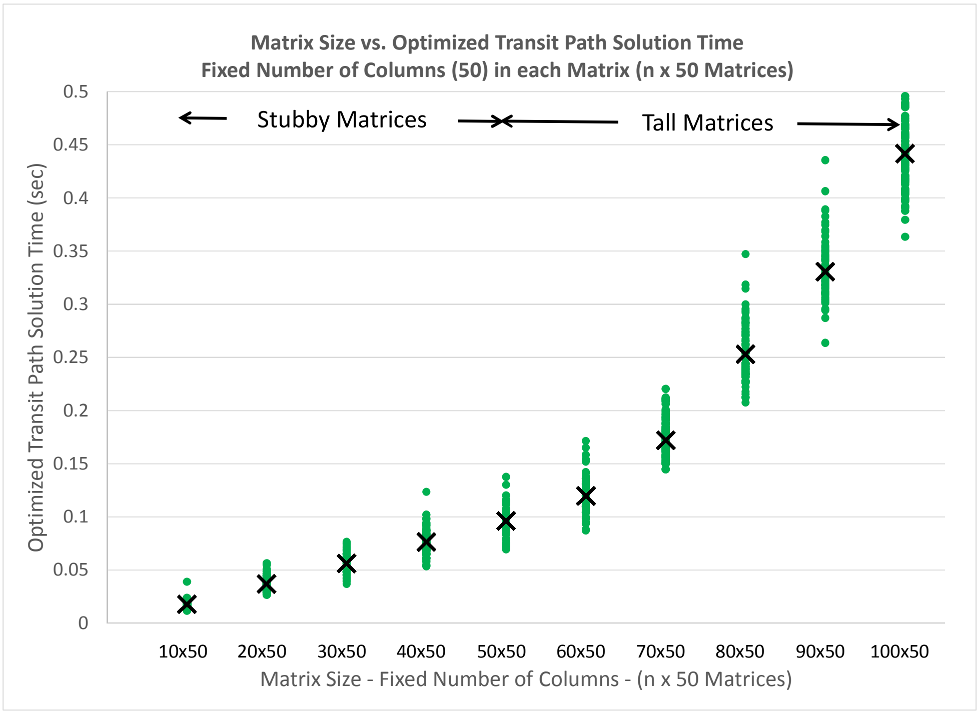


Figure 64: Matrix Size vs. Calculation Time – Fixed # of Columns
(X = Mean Value for Each Matrix Size)

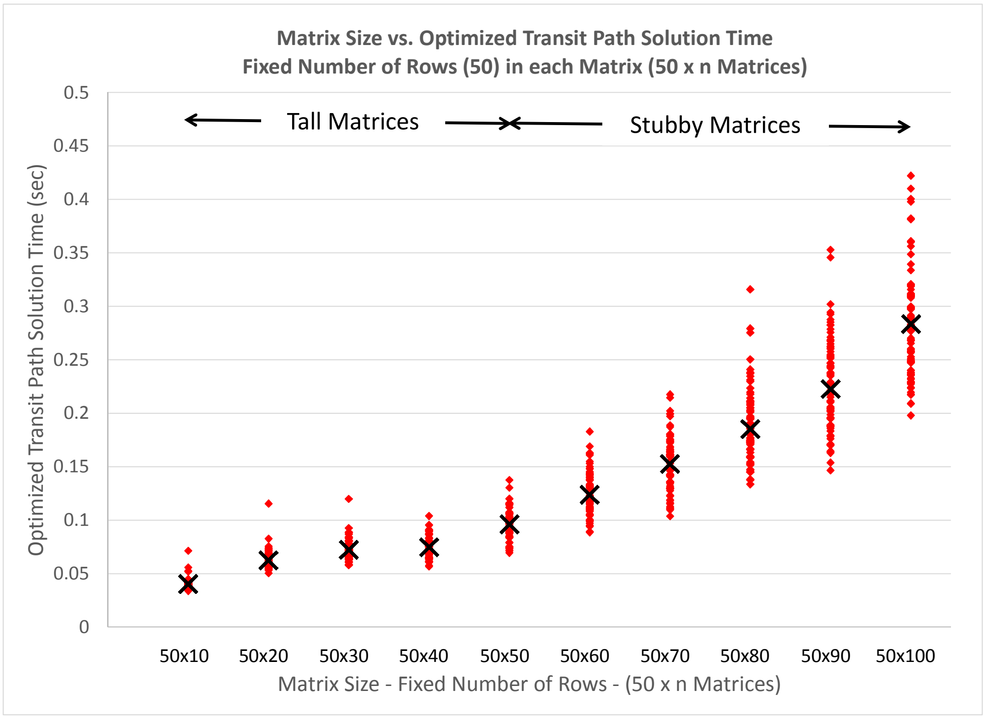


Figure 65: Matrix Size vs. Calculation Time – Fixed # of Rows
(X = Mean Value for Each Matrix Size)

9.5 Comparison of OTP Performance to Other Shortest Path Algorithms

In order to compare the performance of the OTP algorithm to that of the most commonly used shortest path algorithms, one thousand 43x50 matrices were randomly generated and solved with the OTP algorithm. This is the same size matrix as is used in the Arabian Sea piracy problem. From each node within the matrix, except those on the outer edges, travel could occur in eight directions: up/down, left/right, and diagonally. The algorithm was set up to conduct its standard solution methodology, providing the shortest possible path from any point on the "starting line" on one side of the matrix to any point on the "finish line" on the other side of the matrix.

The results of the thousand tests of the OTP algorithm with 43x50 matrices are summarized in Table 7. The thousand test solutions required an average of 386,000 simple calculations, which averaged less than a tenth of a second to complete. The most complicated matrix, with the longest solution time, required 606,000 calculations, which was completed in just under 0.15 seconds.

Table 7: OTP Algorithm Solution Results for One Thousand 43x50 Matrices

	Number of Forward Passes	Number of Backward Passes	Total Solution Time (sec)	Number of Calculations for 43x50 Piracy Matrix
Mean	10.1	10.2	0.0970	386,000
Standard Deviation	2.3	2.1	0.0178	68,600
Maximum Value	18	17	0.1472	606,000
Minimum Value	6	6	0.0633	248,000

The results for the OTP algorithm were then compared to the required number of calculations to solve a 43x50 piracy matrix for some of the most common shortest path algorithms in use today. Each of these algorithms was discussed briefly in Chapter 6. The Dijkstra and Bellman-Ford algorithms are traditionally used to solve single-source shortest path algorithms, giving the shortest path from a specific point in the matrix to all other nodes in the matrix. In order to transform these into a “starting line” to “finish line” type of application, these algorithms would need to be run once for each node on the starting line. In the case of the 43x50 piracy matrix, this would mean that the Dijkstra and Bellman-Ford algorithms would need to be run 43 times in order to evaluate every potential starting point, were a “starting line” solution desired. Both the Floyd-Warshall algorithm and Johnson’s algorithm solve the all-pairs shortest path problem. Because they solve for the shortest path from every node in the matrix to every other node, their solution includes the subset of finding the shortest path from any node on the starting line to any node on the finish line.

Table 8 summarizes the required number of calculations for each of these common shortest path algorithms in order to solve a 43x50 piracy matrix. As discussed in Section 6.2, Dijkstra’s Algorithm implementing a Min-Priority Queue with a Fibonacci Heap is considered the fastest possible solution to the single-source shortest path problem with non-negative edge weights (Cormen et al. 2009). For the 43x50 piracy matrix, this is certainly the case, as its solution requires orders of magnitude fewer calculations than the other methods evaluated. Dijkstra with Fibonacci Heap is especially well suited for applications that have a limited number of connections between nodes, rather than having all nodes interconnected with each other. The piracy

Table 8: Number of Calculations Required to Solve a 43x50 Piracy Matrix for Common Shortest Path Algorithms

Algorithm Name	Route Type	Formula for Number of Calculations	Number of Calculations for 43x50 Piracy Matrix
Traditional Dijkstra	Specific Point to Finish Line	N^2	4,620,000
Traditional Dijkstra	Starting Line to Finish Line	$R * (N^2)$	199,000,000
Dijkstra with Fibonacci Heap	Specific Point to Finish Line	$E + N \log N$	15,500
Dijkstra with Fibonacci Heap	Starting Line to Finish Line	$R * (E + N \log N)$	666,000
Bellman-Ford	Specific Point to Finish Line	$N * E$	17,900,000
Bellman-Ford	Starting Line to Finish Line	$R * (N * E)$	769,000,000
Floyd-Warshall	All Points to All Points	N^3	9,940,000,000
Johnson	All Points to All Points	$N^2 \log N + N * E$	33,300,000

List of Abbreviations	Values for 43x50 Piracy Matrix
N = Number of Nodes in Matrix	N = 43 * 50 = 2,150 nodes
E = Number of Edges in Matrix	E = 8,323 edges
R = Number of Rows in Matrix	R = 43 rows
C = Number of Columns in Matrix	C = 50 columns

matrix is an excellent example of such a matrix, as each of its nodes is only connected to eight other nodes. Thus, it is not surprising that Dijkstra with Fibonacci Heap performs so much better than the other methods. Even if this method were modified to provide the starting line to finish line solution to the 43x50 piracy matrix, by running the application an extra 43 times, it is still significantly faster than all other methods.

Based on the data in Table 7 and Table 8, the Optimized Transit Path algorithm and Dijkstra's Algorithm with a Fibonacci Heap had similar orders of magnitude for the required number of calculations to solve for the shortest path through the piracy 43x50 matrix. Dijkstra with Fibonacci heap will typically be an order of magnitude faster for the single-source problem, while the OTP algorithm will usually be a little faster for the starting line to finish line problem. Both algorithms are significantly faster at solving the problem than the other methods investigated. And both are so fast that, for most modern computers, their solution times for the 43x50 piracy matrix would be almost negligible compared to the overhead time associated with generating and downloading the matrix.

As discussed earlier in Chapter 7, one of the greatest strengths of the Optimized Transit Path algorithm is the additional insight it gives about the cost of diverting from the shortest path. Through its calculation of Optimized Slack, the OTP algorithm provides extra information to the route decision-maker that other methods, including Dijkstra, do not. For the piracy problem, the computation time is similar between the OTP algorithm and Dijkstra with Fibonacci Heap. Thus, because it also provides Optimized Slack data, the OTP algorithm would be a preferred method of solving for the shortest path in the piracy application.

However, the Optimized Transit Path algorithm has a fairly limited set of potential uses outside of the piracy application because it relies on an equal number of connections between nodes – the 8 geographic travel directions between nodes (or 4 if diagonal travel is not permitted). Thus, extensive use of the OTP algorithm only seems applicable to 2-dimensional geographic travel, for finding the shortest path across a flat plane. It is not nearly as versatile of a method as Dijkstra's Algorithm with Fibonacci Heap, which can be used to solve a wider range of shortest path applications.

Another strength of the Optimized Transit Path algorithm for use in the piracy application is that it directly uses an input matrix that is in the same format as the PARS model for piracy prediction. The OTP algorithm uses a matrix in which each node has a cost associated with it. In the algorithm, the net cost of a path through the matrix equals the sum of the costs of the nodes on that path. No transformation is needed of the piracy prediction matrix in order to solve for its shortest path with the OTP algorithm.

To use Dijkstra's Algorithm with Fibonacci Heap for the piracy application, the input matrix must be restructured to match the format required by the algorithm. Dijkstra's method requires a listing of all nodes in the matrix, with a separate matrix listing which nodes are connected to which and what the associated edge costs are between those nodes. Especially because the piracy application always involves the same size of matrix, a computer code could be written to transform the piracy matrix into the format required for Dijkstra's method. However, this does add an extra layer of complexity to solving the problem, though it is certainly not a major difficulty that cannot be overcome.

Chapter 10: Potential Uses in Other Applications

Because it relies on a consistent number of edge connections coming out of each node in the matrix, the Optimized Transit Path algorithm is not foreseen as being useful for computer network or transportation routing problems. It is not versatile enough to handle the variable number of edges associated with each node in these applications. However, it may be well suited for a variety of search applications, especially those involving probability of discovery in a flat plane allowing unlimited motion directions.

10.1 Single Attribute Search

For many applications involving search, especially those in which a matrix map is generated with probability of discovering the lost item, the Optimized Transit Path algorithm could provide a simple and quick method of planning out one's search route. This would be especially true for scenarios in which it is desired to follow a relatively straight path through the matrix, without significant turning or looping, unless it provided significant benefit. Underwater search, in which a towed array sonar is dragged behind the ship, is one such application. Because of the towed array, quick and tight turns are not feasible, so an optimized path across the matrix would be desirable.

In search scenarios, however, the probability matrix usually displays the probability of discovering the missing item, $P_{discovery}$, in each node. A path through the matrix that maximizes $P_{discovery}$ would be desirable. In such cases, a shortest path algorithm such as the OTP algorithm or Dijkstra's algorithm could still be used to find the best search path. This would be done by creating a matrix of probabilities of not discovering the item in each node, $P_{\text{no discovery}}$, then optimizing for the shortest path across this matrix. The shortest path across the $P_{\text{no discovery}}$ matrix would give the path with the lowest net chances of not discovering the missing item. This same path would, in turn, be the most efficient path through the matrix to maximize one's chances of finding the item with a minimal total distance traveled.

For example, suppose the 5x5 matrix shown in Figure 66 represents the probability of finding a missing object at any of 25 different locations. If it were certain that the object was located somewhere within this matrix, then the total probability represented by the sum of all the nodes would be 100%.

2	3	5	2	3
3	6	9	6	4
3	5	7	5	3
6	4	5	3	2
4	3	3	2	2

Note: $P_{\text{discovery}}$ Values are in Percent

Figure 66: Probability of Discovery Matrix for Search Example

If it were desired to travel across this matrix with a minimal distance traveled while maximizing one's chances of finding the missing object, a shortest path algorithm could be used by transforming the matrix into the probability of not finding the object at each node, as shown in Figure 67. The probability of not discovering the object at each node equals 100% minus the probability of finding it at that location: $P_{\text{no discovery}} = 100\% - P_{\text{discovery}}$. The shortest path across Figure 67 would, in turn, represent the shortest path across Figure 66 that had the highest probability of finding the object.

However, in cases like this one, where the probabilities of finding the missing object are very low, the probabilities in the $P_{\text{no discovery}}$ matrix would all be very high numbers. Such high costs at every node would make it unlikely that the optimized path would ever divert from a relatively straight path through the matrix. Any diversion

$$\begin{tikzcd}
98 & 97 & 95 & 98 & 97 \\
97 & 94 & 91 \arrow[r] & 94 \arrow[r] & 96 \\
97 & 95 \arrow[ur] & 93 & 95 & 97 \\
94 \arrow[ur] & 96 & 95 & 97 & 98 \\
96 & 97 & 97 & 98 & 98
\end{tikzcd}$$

Note: $P_{\text{no discovery}}$ Values are in Percent

Figure 67: Probability of No Discovery Matrix for Search Example

upwards or backwards would add a very high cost value to the overall sum, making it unlikely that the optimized path would ever divert that direction. A better option, rather than subtracting $P_{discovery}$ from 100%, as was done in Figure 67, would be to create a modified "No Discovery" matrix with very low, but still positive, values in it. One option for doing this would be to subtract $P_{discovery}$ for each node from the maximum $P_{discovery}$ value in the matrix. For example, in Figure 66, the maximum value of $P_{discovery}$ in the matrix is 9%. Thus, to create the Modified No Discovery matrix, the value of each node was subtracted from 9, with results as shown in Figure 68. The shortest path was then found across this matrix in order to give an optimized path recommendation. Although, in this example for a small x5 matrix, the recommended path remained the same, the optimized path has much more ability to divert from a

$$\begin{tikzcd}
7 & 6 & 4 & 7 & 6 \\
6 & 3 & 0 \arrow[r] & 3 \arrow[r] & 5 \\
6 & 4 \arrow[ur] & 2 & 4 & 6 \\
3 \arrow[ur] & 5 & 4 & 6 & 7 \\
5 & 6 & 6 & 7 & 7
\end{tikzcd}$$

Note: $P_{modified\text{-}no\text{-}discovery}$ Values equal $P_{discovery\text{-}maximum} - P_{discovery}$

Figure 68: Modified No Discovery Matrix for Search Example

completely straight line using the Modified No Discovery matrix than it did with the true $P_{\text{no discovery}}$ matrix.

Because the input required by the Optimized Transit Path algorithm is a matrix of node values in the same format as the probability maps of Figure 66, Figure 67, and Figure 68, it is especially well suited to solving such a search problem.

10.2 Multiple Attribute Search

For applications with multiple attributes of each node (or edge between nodes), each of which it is desirable to optimize, the methods of combining multiple attributes developed in Chapter 5 might have value. In this case, each of the desirable attributes would be expressed in terms of a single, common attribute then combined, either directly or via weighting, into a single value of this attribute for each node in the matrix. The matrix could then be optimized for that common attribute.

For example, in Chapter 5, it was desired to minimize the two attributes of net piracy probability and total transit costs. The piracy probability was transformed into an Expected Monetary Value cost of piracy in each node, which was then combined with transit cost per node to give a single attribute, overall cost for each node. Overall cost could then be optimized for the shortest possible path through the matrix. A similar methodology might be useful in other applications besides piracy.

10.2.1 Underwater Search Applications

Many aspects of underwater search, as discussed in Section 4.3, are similar to the piracy avoidance problem. The most similar feature is the use of a matrix with probability values to represent an area. As with piracy prediction, each node in the underwater search matrix represents an equally sized geographic area of the sea. The value of each node represents the probability that the missing object is located within the area represented by that node. Another similarity is that, in underwater search, there is a cost associated with transiting each node. It is very expensive to conduct an underwater search with towed array sonar, costing many thousands of dollars each day, so each node passed through in the search probability matrix will have a transit cost associated with it.

However, the underwater search problem has many differences from piracy avoidance as well. First, in underwater search, the goal of transiting the search area is to maximize one's probability of locating the missing object while minimizing transit costs. In the piracy application, the goal was to minimize both the total probability of a pirate encounter and the transit cost simultaneously. Second, in underwater search, the plan is typically to transit the matrix area multiple times until either the missing object is found or the allocated time and funds for the search have been exhausted. To this end, after each node of the matrix has been unsuccessfully searched, the probability for that node is decreased, though usually not to zero, according to Bayes' Theorem. Likewise, the unsuccessful search of one node results in a small increase in the probability associated with every other node in the matrix. Thus, the values in the probability matrix are constantly changing. In the piracy application, because there is never a fixed nor a known number of pirates in the Arabian Sea at a given time, the absence of pirate activity in one node of the matrix does not mean that the probability of pirate activity will be increased in the other nodes. The PARS piracy probability matrix is updated every twelve hours, while the underwater search probability matrix can be updated constantly.

Despite the differences, a similar approach could be used to solve the optimized underwater search problem as was used to solve the piracy problem. The methods of combining multiple node attributes (Chapter 5) could be used along with the Optimized Transit Path algorithm (Chapter 7) and Bayes' Theorem (Chapter 4) to provide an optimized plan for underwater search. This plan would be one that maximizes the chances of finding the missing object over the course of multiple passes through the matrix area, while minimizing the costs associated with such a search.

Underwater search has two attributes that it is desired to optimize: maximum probability of finding the missing object and minimum transit cost. To combine these two attributes, the probability of finding the missing object could be transformed into a cost value, from which the transit cost of searching that node would be subtracted.

$$EMV_{\text{search per node}} = P_{\text{object in node}} * P_{\text{discovery}} * \text{Benefit}_{\text{object discovered}} - C_{\text{transit}} \qquad (38)$$

The Expected Monetary Value for each node equals the product of three terms – the probability that the missing object is located in that area, the probability that the missing object would be detected by a search of that area if it were located there, and the financial benefit that would be realized if the missing object were found - from which is subtracted the cost associated with transiting through and conducting the search in that node. In order to find the optimized path through the matrix, the value of each node would first be transformed into an $\text{EMV}_{\text{search per node}}$ value.

Note that $\text{EMV}_{\text{search per node}}$ could be either a positive or a negative number, so the matrix may include both. For low probability locations, the cost of transiting and searching the location could outweigh the benefit of searching that area, resulting in a negative number. Meanwhile, in high probability locations, the financial benefit of a search would hopefully be greater than the search costs, resulting in positive values.

Both the Optimized Transit Path algorithm and Dijkstra's algorithm find the shortest path across a matrix. To use either algorithm to find the optimized path across the $\text{EMV}_{\text{search per node}}$ matrix, the matrix would need to be transformed again into a modified matrix related to costs of not finding the object, in the same fashion as was described in section 9.1 for a single attribute probability matrix. In this case, the EMV value for each node would be subtracted from the maximum EMV value in the matrix to get the Modified No Discovery matrix. Either the OTP algorithm or Dijkstra's algorithm could then be used to determine the optimized path across the algorithm.

As each node is unsuccessfully searched, its probability value decreases, also decreasing its EMV but increasing its node value in the Modified No Discovery matrix. Likewise, the probability of every other node in the matrix would increase slightly, also increasing its EMV and decreasing its node value in the Modified No Discovery matrix. Because the OTP and Dijkstra calculations are performed so quickly, the optimized path recommendation could be updated almost instantaneously with each unsuccessful search of a node. Such path calculations from within the matrix would be based on starting from a specified point in the matrix, the search vessel's current location. Once the end of the matrix was reached, a new optimized path back through the matrix in the opposite direction would be calculated, the vessel would turn around, and the search would continue.

For a scenario such as the underwater search application, the entire search plan could be determined ahead of time. As long as transit and search costs, an initial probability matrix for the missing object's location, and probability of detection information were available ahead of time, the search plan could be created ahead of time in order to maximize the overall probability of finding the missing object while keeping the total transit costs at or below a specified value.

10.2.2 Other Multiple Attribute Applications

For applications with multiple attributes for each node or edge, it is often desired to optimize more than one of these attributes at a time. In such cases, the idea of combining multiple attributes into a single common attribute, introduced in Chapter 5, may be have value, whether by directly combining the attributes or weighting them based on importance.

A simple example of such an application would be related to transportation routing and traffic avoidance. Two attributes that one might want to minimize would include typical driving time and amount of traffic encountered on a particular road. As is commonly done, these two attributes could be combined by expressing them both in terms of time – no traffic driving time for the road and amount of delay time associated with traffic – then combining them to give overall driving time for the road. Overall driving time would then be optimized for a shortest path route between two points.

However, the idea of weighting one or more of the attributes, as discussed in Chapter 5, could have value to this application. Based on a driver's priorities and preferences, the traffic routing algorithm might apply a weighting factor to the traffic time portion of the overall time. For a driver just interested in getting to his destination as quickly as possible, the weighting between typical driving time and traffic time would be equal, and the optimized route would be the one that gets him to his destination most quickly. However, the driver might be given an "I hate traffic! I just want to keep moving, even if it takes longer." weighting option, in which the time spent waiting in traffic was more heavily weighted than the regular driving time. A weighted, combined sum of times would populate each node or edge in the matrix of overall drive times, for which a shortest path route would be found. This shortest path would not necessarily be the absolutely quickest drive time option. It also wouldn't necessarily guarantee that all traffic was avoided, depending on the level of priority placed on traffic avoidance. But it would provide a path that avoids most traffic, keeps the driver's car moving as much as possible, and still gets him to his destination in a reasonable amount of time.

Other applications, in which it is desirable to optimize multiple attributes of a node or edge, could certainly take advantage of these methods. First, combine multiple attributes into a single, common attribute to be optimized. Next, employ a weighting factor to allow some attributes to be more heavily prioritized than the others.

Chapter 11: Conclusions and Future Work

Despite a lowered number of incidents in the past two years, maritime piracy in the Arabian Sea remains a real problem that will not go completely away any time soon. Mostly due to large amounts of military interdiction by multinational forces, the number of pirate attacks in the region has dropped dramatically. However, as long as the factors of geographic location, lawless government, and extreme poverty remain in Somalia, piracy will likely flourish again in the Arabian Sea once enforcement efforts wane. Meanwhile, off the coast of Nigeria, piracy in West Africa has grown tremendously in the past five years, due to mostly the same reasons as it will thrive again in Somalia.

The methods developed in this paper allow a mariner to take maximum advantage of the piracy avoidance tools that might be available to him, most notably the PARS piracy prediction matrix. By taking into account the specific characteristics of an individual ship and its counter-piracy measures, an optimized route through the Arabian Sea can be calculated for each vessel. Based on a mariner's priorities, this route can be optimized for minimum probability of a pirate encounter, minimum overall cost, or a weighted combination of the two. This ability to see the financial impact of his choices allows a mariner to select the optimized route that is best for him.

As part of this research, a new algorithm was developed for finding the shortest path from one side of a two-dimensional matrix array to the other, the Optimized Transit Path algorithm. The OTP algorithm uses a matrix input in the same format as that provided by the PARS model, with each node of the matrix representing an equally sized geographic area and the value of the node representing the attribute to be optimized. In addition to the simplicity and speed of its calculations, one of the most useful features of the OTP algorithm is its calculation of Optimized Slack. Nodes with OS values of zero are on the shortest path through the matrix. Non-zero values of OS indicate the penalty, in terms of increased path length, that would be incurred if a point were diverted to in the most efficient manner possible. Such information could also be useful to a mariner in his route decisions.

A computer code was written of the Optimized Transit Path algorithm, allowing extensive experimental testing to be done on the algorithm. These results were used both to characterize and understand the behavior of the OTP algorithm and to compare its performance to that of the shortest path algorithms that are most commonly used today. Based on this testing, it was found that the OTP algorithm required a similar number of calculations to solve the piracy matrix as the fastest of the common algorithms, Dijkstra's Algorithm implementing a Min-Priority Queue with a Fibonacci Heap. Both the OTP algorithm and Dijkstra's Algorithm with Fibonacci Heap were multiple orders of magnitude faster at solving the piracy matrix than any of the other algorithms investigated.

Because it is able to use the piracy prediction matrix directly as an input, the Optimized Transit Path algorithm is especially well suited for use in solving the piracy problem. More importantly, its calculation of Optimized Slack provides an insight into the route decision-making process that other methods do not. However, the use of the OTP algorithm is fairly limited, as it is only well suited for matrices that represent a flat plane of interconnected geographic areas, with movement from a node limited only to the eight adjacent nodes surrounding it. Dijkstra's Algorithm with Fibonacci Heap, meanwhile, requires significant alteration of the format of the piracy prediction matrix in order to calculate its solution, and it does not provide awareness of the additional cost of diverting from the shortest path. However, the Dijkstra method is far more versatile, as it allows any number of edges between nodes in the matrix. Hence the reason it is used as the backbone for the majority of shortest path applications today.

Future work, building on the methods in this paper, would include piracy prediction and route optimization in other parts of the world, user-friendly GUI output of the optimized route through a geographic region, other applications with multiple attributes to simultaneously optimize, and other two-dimensional, geographic routing problems involving matrix distributions. One other application where these methods seem especially promising and useful is within the field of underwater search. In much the same way that it can quickly identify optimized routes for multiple parameters of the piracy problem and comparison between them, the Optimized Transit Path algorithm could likewise provide similar path optimization based on multiple parameters of the underwater search problem.

Appendices

Appendix 1: MATLAB Code for Optimized Transit Path Algorithm

```
%%

%File Name: Optimized_Transit_Path_Algorithm_Read_External_File_Input

clear; clc


%INPUT AND PREPARE THE MATRIX TO BE OPTIMIZED

%Input the matrix that an Optimized Transit Path is desired for.
%Read the matrix from an external source, in this case an Excel file.

run Read_matrix_from_external_file_source


%Start a timer so the overall computation time can be tracked.

tic;


%Create Rows along the top and bottom of the original matrix with very high values
%to act as a boundary that the Optimized Transit Path will not ever cross into.
%Also, count the number of rows, a, number of columns, b, and number of nodes in
%the matrix.

run Bound_original_matrix


%Set initial values of Minimum Start (MS) and Minimum Finish (MF) to be very
%high. These will be lowered through successive forward passes.

run Prepare_initial_MS_and_MF_matrices


%FORWARD PASS CALCULATIONS

%Conduct a forward pass of calculations for the Minimum Start and Minimum Finish
%values for each node, with a methodology analogous to forward pass scheduling
%techniques of the Critical Path Method.  However, because it allows backtracking
%and vertical motion, the forward pass process for finding the Optimized Transit
%Path will be an iterative one that may require multiple passes.  Each pass will build
%upon the values from the previous path.
```

```
%Use a "for" loop to allow forward passes to occur until a solution is reached.  Set
%the "for" loop to repeat up to number_of_nodes times, a number which will never
%be reached.  An "if" statement later in the code, along with a "break" statement,
%will stop the code from looping once the Optimized Transit Path has been found,
%rather than running all of the potential loops.

for number_of_forward_loops = 1:number_of_nodes


%Set up confirmation matrices, Previous_MS and Previous_MF, equal to the
%previous MS and MF values.  At the end of each "for" loop, these will be compared
%to the newly calculated MS and MF values.  If, at the end of the loop,
%MS = Previous_MS and MF = Previous_MF, then no further changes to MS and
%MF are possible, the Optimized Transit Path values have been found, and the "for"
%loop can stop.

Previous_MS = MS;
Previous_MF = MF;


%Conduct forward pass calculations to calculate updated MS and MF values.

 run Forward_pass_calculations


%If no changes were made during a forward pass compared to the previous forward
%pass, as confirmed by MF equaling Previous_MF and by MS equaling
%Previous_MS, then the MS and MF values of the Optimized Transit Path have
%been found and the "for" loop can stop.

%If any changes were made to MS and MF compared to the previous forward
%pass, then at least one additional forward pass must be completed to
%find the Optimized Transit Path.

%To save calculation time, if MF equals Previous_MF, then the forward pass loop
%will stop.  A "break" command is inserted within an "if" statement to stop the
%program from running all of its loops if this has been achieved.

Stop_Loop_Check = MF - Previous_MF;
if abs(Stop_Loop_Check) < 0.00001
   break
end


%If MF does not equal Previous_MF, then another series of forward passes will be
```

```
%conducted up to number_of_forward_loops times.  The below "end" command
%cycles back to the start of the number_of_forward_loops "for" loop.

 end


%Count the number of forward passes needed to arrive at the Optimized Transit Path
%solution and the time it took to reach the solution.

run Forward_pass_timer


%BACKWARD PASS CALCULATIONS

%To more easily identify which nodes are on the Optimized Transit Path, a backward
%pass is conducted to identify the Required Finish (RF) and Required Start (RS)
%values for each node.

%Set initial values of Required Start (RS) and Required Finish (RF) to be very high.
%These will be lowered through successive backward passes.

run Prepare_initial_RS_and_RF_matrices


%Use a "for" loop to allow backward passes to occur until a solution is reached.  Set
%the "for" loop to repeat up to number_of_nodes times, a number which will never
%be reached. An "if" statement later in the code, along with a "break" statement,
%will stop the code from looping once the Optimized Transit Path has been found,
%rather than running all of the potential loops.

 for number_of_backward_loops = 1:number_of_nodes


%Set up confirmation matrices, Previous_RS and Previous_RF, equal to the previous
%RS and RF values.  At the end of each "for" loop, these will be compared to the
%newly calculated RS and RF values.  If, at the end of the loop, RS = Previous_RS
%and RF = Previous_RF, then no further changes to RS and RF are possible, the
%Optimized Transit Path values have been found, and the "for" loop can stop.

Previous_RS = RS
Previous_RF = RF


%Begin backward pass calculations

run Backward_pass_calculations
```

```
%If no changes were made during a backward pass compared to the previous
%backward pass, as confirmed by RF equaling Previous_RF and by RS equaling
%Previous_RS, then the RS and RF values of the Optimized Transit Path have
%been found and the "for" loop can stop.

%If any changes were made to RS and RF compared to the previous backward
%pass, then at least one additional backward pass must be completed to find the
%Optimized Transit Path.

%To save calculation time, if RF equals Previous_RF, then the backward pass
%loop will stop.  A "break" command is inserted within an "if" statement to stop the
%program from running all of its loops if this has been achieved.

Stop_Loop_Check = RS - Previous_RS;
if abs(Stop_Loop_Check) < 0.00001
   break
end


%If RS does not equal RS_Confirm, then another series of backward passes will be
%conducted up to number_of_backward_loops times. The below "end" command
%cycles back to the start of the number_of_backward_loops "for" loop.

 end


%Count the number of backward passes needed to arrive at the Optimized Transit
%Path solution and the time it took to reach the solution.

run Backward_pass_timer


%OPTIMIZED SLACK CALCULATIONS TO IDENTIFY OPTIMIZED TRANSIT
PATH

%Optimized Slack (OS) is used to identify the Optimized Transit Path through the
%matrix.  This methodology is analogous to how Total Slack is used in CPM to find
%the critical path through a network diagram.

%Optimized Slack is the difference between Required Finish (RF) and Minimum
%Finish (MF) for a node. Optimized Slack also equals the difference between
%Required Start (RS) and Minimum Start (MS) for a node. Relationships to find
%Optimized Slack:  OS = RF - MF   or   OS = RS - MS
```

```
%Those nodes with an Optimized Slack of zero are the nodes on the Optimized
%Transit Path through the matrix.

run Optimized_Slack_calculation


%Strip the bounding rows from the MS, MF, RS, RF, and OS matrices.

run Simplify_optimized_matrices


%Measure the total time spent on all calculations since the beginning.

run Final_timer


%Present the final results to the user in a concise manner. Final results include the
%Optimized Slack matrix plus statistics about the solution time and number of
%passes.

run Final_results_summary


%Write the Optimized Slack matrix to Excel for visual presentation

run Write_OS_solution_to_excel
```

Appendix 1A: MATLAB Subroutine to Read Matrix Input from External File

```
%File Name: Read_matrix_from_external_file_source


%Read the input matrix from an external file that contains the matrix

Matrix_input = xlsread('External_Matrix_Source.xlsx')
```

Appendix 1B: MATLAB Subroutine to Bound Matrix with High Value Rows

```
%File Name: Bound_original_matrix

%Define a and b as the number of rows and columns, respectively, in the
%original input matrix.

[a b] = size(Matrix_input);


%Determine the number of nodes in the matrix.

number_of_nodes = a * b;


%Create Rows along the top and bottom of the original matrix with very
%high values to act as a boundary that the optimized path will not ever
%want to cross into.

High_Value = 1000000000;
Bounding_Row = High_Value * ones(1,b);

Bounded_Matrix_input = [Bounding_Row; Matrix_input; Bounding_Row];
```

Appendix 1C: MATLAB Subroutine to Prepare Initial MS and MF Matrices

```
%File Name: Prepare_initial_MS_and_MF_matrices


%Set initial values of Minimum Start (MS) and Minimum Finish (MF) to be very
%high. These will be lowered through successive forward passes.

MF = ones(a+2,1) * Bounding_Row;
MS = MF;


%Set the values in the first column of the Minimum Start (MS) matrix to zero, except
%for the values in the bounding rows, which will each stay as a very high number to
%keep the optimized path contained to the original matrix.

for i = 2:b+1           %All rows except top and bottom bounding rows
 MS(i,1) = 0;
 end
```

Appendix 1D: MATLAB Subroutine to Conduct Forward Pass Calculations

```
%File Name: Forward_pass_calculations


%Begin forward pass calculations.


%Calculate MS and MF values for column 1.
%Minimum Finish (MF) for each node in column 1 equals its Minimum Start (MS),
%which is zero for column 1, plus the node's cost value.

 for i = 2:a+1                        %All rows except top and bottom bounding rows
    MF(i,1) = MS(i,1) + Bounded_Matrix_input(i,1);          %Column 1 values
 end


%Conduct a forward pass to determine the new MS and MF values for all remaining
%columns except the last column.

for j = 2:b-1                         %All columns except the first and last

%Initially, set the Minimum Start (MS) for each node as the smallest of the
%Minimum Finish (MF) values of the 3 nodes in the previous column feeding into it.

 for i = 2:a+1                        %All rows except top and bottom bounding rows
    Prev_MF = [MF(i,j-1) MF(i+1,j-1) MF(i-1,j-1)];
    MS(i,j) = min(Prev_MF);
 end


%Next, check whether a path going vertically up or down or else a path that
%backtracks would have resulted in a lower MS value.  If so, replace the MS value
%for each node with these values.

for i = 2:a+1                         %All rows except top and bottom bounding rows

    %Check if vertically downward Minimum Finish (MF) values are lower
    %than the current Minimum Start (MS) value.  If so, replace MS.

    if MF(i+1,j) < MS(i,j)
       MS(i,j) = MF(i+1,j);
    end
    %Check if vertically upward Minimum Finish (MF) values are lower
    %than the current Minimum Start (MS) value.  If so, replace MS.

    if MF(i-1,j) < MS(i,j)
       MS(i,j) = MF(i-1,j);
    end


    %Check if backtracking Minimum Finish (MF) values are lower
    %than the current Minimum Start (MS) values.  If so, replace MS.

    %Check reverse and horizontal MF value.

    if MF(i,j+1) < MS(i,j)
       MS(i,j) = MF(i,j+1);
    end


    %Check reverse and downward MF value.
    if MF(i+1,j+1) < MS(i,j)
       MS(i,j) = MF(i+1,j+1);
    end


    %Check reverse and upward MF value.
    if MF(i-1,j+1) < MS(i,j)
       MS(i,j) = MF(i-1,j+1);
    end

%End the calculation for all of the nodes in the column from the "for" loop of
%i values begun earlier (for i = 2:a+1).

 end


%Now that all of the Minimum Start (MS) values have been set, recalculate
%the Minimum Finish (MF) values for each node in the column.
%MF = MS + Node Cost

for i = 2:a+1                          %All rows except top and bottom bounding rows
   MF(i,j) = MS(i,j) + Bounded_Matrix_input(i,j);
end
```

```
%End the calculation for all of the columns except the last one from the
%"for" loop of j values begun earlier (for j = 2:b-1).

end


%Conduct calculations for the last column in a similar manner to the others.
%However, there is no need to conduct checks for going vertically or backwards
%because once the last column has been reached, the path is completed.  Adding
%more steps would just add more cost to the path.

for j = b                              %last column


 %The Minimum Start (MS) for each node is the smallest of the Minimum
 %Finish (MF) values of the 3 nodes in the previous column feeding into it.

 for i = 2:a+1                         %All rows except top and bottom bounding rows
    Prev_MF = [MF(i,j-1) MF(i+1,j-1) MF(i-1,j-1)];
    MS(i,j) = min(Prev_MF);
 end


%Now that all of the Minimum Start (MS) values have been set for the last
%column, recalculate the Minimum Finish (MF) values for each node.
%MF = MS + Node Cost

 for i = 2:a+1                         %All rows except top and bottom bounding rows
    MF(i,j) = MS(i,j) + Bounded_Matrix_input(i,j);
 end


%End calculations for the final column
end


%This round of forward pass calculations for the Minimum Start (MS) and
%Minimum Finish (MF) values for each node is now complete.
```

Appendix 1E: MATLAB Subroutine to Time and Count Forward Passes

```
%File Name: Forward_pass_timer


%Count the number of forward passes needed to arrive at the Optimized Transit Path
%solution.

number_of_forward_passes = number_of_forward_loops;


%Record the time it took to determine the Optimized Transit Path solution via
%forward pass calculations.

time_for_forward_pass = toc;
```

Appendix 1F: MATLAB Subroutine to Prepare Initial RS and RF Matrices

```
%File Name: Prepare_initial_RS_and_RF_matrices


%Set initial values of Required Start (RS) and Required Finish (MF) to be large
%negative numbers. These will be increased through successive backward passes.
%Values in the last column of the Required Finish (RF) matrix are set equal to the
%Minimum Finish of the last node on the Optimized Transit Path, which was found
%in the final forward pass.

%Set initial values of Required Start (RS) and Required Finish (RF) to be very large
%negative numbers.

Negative_Bounding_Row = Bounding_Row * -1;
RF = ones(a+2,1) * Negative_Bounding_Row;
RS = RF;


%Set the values in the last column of the Required Finish (RF) matrix to equal the
%Minimum Finish value of the last node in the Optimized Transit Path, as calculated
%from the final forward pass.  However, the values in the bounding rows will stay as
%a very high number to keep the Optimized Transit Path contained to the original
%matrix.

Optimized_Path_Minimum_Finish = min(MF(:,b));

 for i = 2:a+1
 RF(i,b) = Optimized_Path_Minimum_Finish;
 end
```

Appendix 1G: MATLAB Subroutine to Conduct Backward Pass Calculations

```
%File Name: Backward_pass_calculations


%Determine Required Finish (RF) and Required Start (RS) values of nodes in the
%final (right-most) column.  RF of final column nodes was already set equal to the
%Minimum Finish of the last node on the Optimized Transit Path forward through
%the matrix.  RS of each node equals its Required Finish (RF) minus the node's cost
%value from the original input matrix.

 for i = 2:a+1                          %All rows except top and bottom bounding rows
    RS(i,b) = RF(i,b) - Bounded_Matrix_input(i,b);              %Final column

 end


%Conduct a backward pass of calculations to determine the RF and RS values for all
%columns except the first column and the final column.

for j = 1:b-2               %Loop that repeats number of columns minus two times


%Initially, set the Required Finish (RF) for each node as the smallest of the Required
%Start (RS) values of the 3 nodes in the column feeding into it (the column
%immediately to its right).

for i = 2:a+1                          %All rows except top and bottom bounding rows
    Prev_RS = [RS(i,b-j+1) RS(i+1,b-j+1) RS(i-1,b-j+1)];
    RF(i,b-j) = max(Prev_RS);
end


%Next, check whether a path going vertically up or down or else a path that
%backtracks would have resulted in a lower RF value.  If so, replace the RF value for
%each node with these values.

for i = 2:a+1                          %All rows except top and bottom bounding rows

    %Check if vertically upward Required Start (RS) values are greater
    %than the current Required Finish (RF) value.  If so, replace RF.

    if RS(i+1,b-j) > RF(i,b-j)
       RF(i,b-j) = RS(i+1,b-j);
    end
    %Check if vertically downward Required Start (RS) values are greater
    %than the current Required Finish (RF) value.  If so, replace RF.

    if RS(i-1,b-j) > RF(i,b-j)
       RF(i,b-j) = RS(i-1,b-j);
    end


    %Check if backtracking Required Start (RS) values are greater
    %than the current Required Finish (RF) values.  If so, replace RF.

    %Check reverse and up RS value
.
    if RS(i,b-j-1) > RF(i,b-j)
       RF(i,b-j) = RS(i,b-j-1);
    end


    %Check reverse and horizontal RS value.

    if RS(i+1,b-j-1) > RF(i,b-j)
       RF(i,b-j) = RS(i+1,b-j-1);
    end


    %Check reverse and down RS value.

    if RS(i-1,b-j-1) > RF(i,b-j)
       RF(i,b-j) = RS(i-1,b-j-1);
    end


%End the calculation for all of the nodes in the column from the "for" loop of
%i values begun earlier (for i = 2:a+1).

 end


%Now that all of the Required Finish (RF) values have been set, recalculate the
%Required Start (RS) values for each node in the column.  RS = RF - Node Cost

for i = 2:a+1                    %All rows except top and bottom bounding rows
   RS(i,b-j) = RF(i,b-j) - Bounded_Matrix_input(i,b-j);
end
```

```
%End the calculation for all of the columns except the first column from the
%"for" loop of j values begun earlier (for j = 1:b-2).

end


%Conduct calculations for the first column in a similar manner to the others.
%However, there is no need to conduct checks for going vertically or backwards
%because once the first column has been reached, the path is completed.  Adding
%more steps would just add cost to the path.

for j = 1


%The Required Finish (RF) for each node in the first column is the greatest of the
%Required Start (RS) values of the 3 nodes in the column feeding into it (column 2).

for i = 2:a+1                    %All rows except top and bottom bounding rows
    Prev_RS = [RS(i,j+1) RS(i+1,j+1) RS(i-1,j+1)];
    RF(i,j) = max(Prev_RS);
end


%Now that all of the Required Finish (RF) values have been set for the first column,
%recalculate the Required Start (RS) values for each node in the column.
%RS = RF - Node Cost

for i = 2:a+1                    %All rows except top and bottom bounding rows
    RS(i,j) = RF(i,j) - Bounded_Matrix_input(i,j);
end

%End calculations for the first column
end


%This round of backward pass calculations for the Required Start (RS) and
%Required Finish (RF) values for each node is now complete.
```

Appendix 1H: MATLAB Subroutine to Time and Count Backward Passes

```
%File Name: Backward_pass_timer


%Count the number of backward passes needed to arrive at the Optimized Transit
%Path solution.

number_of_backward_passes = number_of_backward_loops;


%To calculate the time it took to determine Optimized Transit Path solution via
%backward passes, subtract the forward pass time from the total elapsed time

time_for_backward_pass = toc - time_for_forward_pass;
```

Appendix 1I: MATLAB Subroutine to Calculate Optimized Slack

```
%File Name: Optimized_Slack_calculation


%Optimized Slack (OS) is used to identify the Optimized Transit Path through the
%matrix.  This methodology is analogous to how Total Slack is used in CPM to find
%the critical path through a network diagram.

%Optimized Slack is the difference between Required Finish (RF) and Minimum
%Finish (MF) for a node. Optimized Slack also equals the difference between
%Required Start (RS) and Minimum Start (MS) for a node.
%Relationships to find Optimized Slack:  OS = RF - MF   or   OS = RS - MS

%Those nodes with an Optimized Slack of zero are the nodes on the Optimized
%Transit Path through the matrix.

OS = RF - MF;
```

Appendix 1J: MATLAB Subroutine to Simplify the Optimized Matrices

```
%File Name: Simplify_optimized_matrices


%Present the optimized transit path MS, MF, RS, RF, and OS matrices without the
%bounding rows.

MS_Optimized_Transit_Path = MS(2:a+1,:);

MF_Optimized_Transit_Path = MF(2:a+1,:);

RS_Optimized_Transit_Path = RS(2:a+1,:);

RF_Optimized_Transit_Path = RF(2:a+1,:);

OS_Optimized_Transit_Path = OS(2:a+1,:);
```

Appendix 1K: MATLAB Subroutine to Record the Time for All Calculations

```
%File Name: Final_timer


%Measure the total time spent on all calculations since the beginning of the process,
%not including time spent to import the matrix.

time_for_all_calculations = toc;
```

Appendix 1L: MATLAB Subroutine to Present the Final Results

```
%File Name: Final_results_summary


%Present the final results to the user in a concise manner.
%Final results include the Optimized Slack matrix plus statistics about the solution
%time and number of passes.

%Display the Optimized Slack matrix for the Optimized Transit Path.

OS_Optimized_Transit_Path


%Add verbiage instructing the user how to interpret Optimized Slack values to
%determine the Optimized Transit Path.

fprintf('The Optimized Transit Path for minimum total cost follows those nodes\n')
fprintf('with Optimized Slack (OS) values of zero in the above matrix,
OS_Optimized_Transit_Path.\n\n')


fprintf('Non-zero, negative values of Optimized Slack indicate the penalty
associated\n')
fprintf('with diverting to that node in terms of increase in the total cost.\n\n')


%Display statistics about the overall solution time, as well as about the solution time
%and number of passes for the forward pass and backward pass calculations.

fprintf('Arriving at the optimized solution took %3.4f
seconds.\n',time_for_all_calculations)

fprintf('Of this total time, %3.4f seconds was spent completing %d forward
passes.\n',time_for_forward_pass, number_of_forward_passes)

fprintf('Of this total time, %3.4f seconds was spent completing %d backward
passes.\n',time_for_backward_pass, number_of_backward_passes)
```

Appendix 1M: MATLAB Subroutine to Write Solution to External Source

```
%File Name: Write_OS_solution_to_excel


%Copy the original input matrix to Excel for ease of visual presentation, in case the
%matrix did not originate from Excel.

xlswrite('Input_Matrix.xls',Matrix_input)


%Copy the Optimized Slack matrix for the Optimized Transit Path solution to Excel
%for ease of visual presentation.

xlswrite('OS_Values_for_Optimized_Transit_Path.xls',OS_Optimized_Transit_Path)
```

Bibliography

Abraham, I., Fiat, A., Goldberg, A., and Werneck, R. 2012. “Highway Dimension, Shortest Paths, and Provably Efficient Algorithms.” ACM-SIAM Symposium on Discrete Algorithms. http://research.microsoft.com/pubs/115272/soda10.pdf

Archibugi, D., and M. Chiarugi. 2009. “Piracy Challenges Global Governance.” http://www. Opendemocracy. Net/article/piracy-challenges-global-governance

Bang-Jensen, J. and Gutin, G. 2009. Digraphs: Theory, Algorithms and Applications, 2nd edition. London UK. Springer-Verlag.

Bateman, S. 2012. “Killing Pirates: Dilemma of Counter-piracy.” http://dr.ntu.edu.sg/handle/10220/7579.

Berger, J. 1985. Statistical Decision Theory and Bayesian Analysis. New York, NY: Springer-Verlag.

Bhandari, R. 1999. Survivable Networks: Algorithms for Diverse Routing. New York, NY. Springer-Verlag.

Bellman, R. 1958. "On a Routing Problem." Quarterly of Applied Mathematics 16. https://www.ams.org/mathscinet-getitem?mr=0102435

Black, P. 2004. Dictionary of Algorithms and Data Structures. National Institute of Standards and Technology. http://www.nist.gov/dads/HTML/johnsonsAlgorithm.html

Bridger, James. 2011. “The Atlantic Council of Canada » Somali Piracy and the World’s Response.” http://atlantic-council.ca/somali-piracy-and-the-worlds-response/.

Chalk, P. 2008. The Maritime Dimension of International Security: Terrorism, Piracy, and Challenges for the United States. Vol. 697. Rand Corporation.

Chalk, P. 2009. Maritime Piracy: Reasons, Dangers and Solutions. DTIC Document. http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA493656.

Chen, M., Chowdhury, R., Ramachandran, V., Roche, D., and Tong, L. 2007. "Priority Queues and Dijkstra's Algorithm - UTCS Technical Report TR-07-54 - 12 October 2007." Austin, TX. The University of Texas at Austin, Department of Computer Sciences.

Cho, J. 2003. Shortest Path Problems in a Stochastic and Dynamic Environment. Wright-Patterson Air Force Base, OH. Air Force Institute of Technology.

Cisco. 2014. Internetworking Technology Handbook. http://www.cisco.com/c/en/us/td/docs/internetworking/technology/handbook/ito_doc.html

Cormen, T., Leiserson, C., and Rivest, R. 2009. Introduction to Algorithms, 3rd edition. Cambridge, MA. MIT Press.

DeGroot, M. 2005. Optimal Statistical Decisions. Hoboken, NJ: Wiley.

Dijkstra, E. 1959. "A Note on Two Problems in Connection with Graphs." Numerische Mathematik 1. http://www.m3.ma.tum.de/twiki/pub/MN0506/WebHome/dijkstra.pdf

Dillon, D. R. 2005. “Maritime Piracy: Defining the Problem.” SAIS Review 25 (1): 155–165.

Duda, D., and T. Szubrycht. 2009. “The Somali Piracy New or Old Challenge for International Community.” Marine Navigation and Safety of Sea Transportation. CRC Press, London: 743–750.

Flament, M. 2005. "ActMAP Final Report." ActMAP Consortium. http://www.transport-research.info/Upload/Documents/201007/ 20100726_152452_50550_ActMAP-final%20report.pdf

Floyd, R. 1962. "Algorithm 97: Shortest Path." Communications of the ACM 5. http://dx.doi.org/10.1145%2F367766.368168

Fredman, M. and Tarjan, R. 1984. "Fibonacci Heaps and Their Uses in Improved Network Optimization Algorithms." 25th Annual Symposium on Foundations of Computer Science. IEEE. http://dx.doi.org/10.1109%2FSFCS.1984.715934

Galletti, S.C. 2007. “Old and New Threats: Piracy and Maritime Terrorism.” EuroCrime. http://www. Southchinasea. org/docs/Galletti-Piracy,% 20Old% 20and% 20New% 20Threats. Pdf.

Gardner, F. 2012. "Seeking Somali Pirates, From the Air." http://www.bbc.co.uk/news/world-middle-east-17095887

Geopolicity Inc. 2011. “The Economics of Piracy: Pirate Ransoms & Livelihoods Off the Coast of Somalia”. Geopolicity. http://www.geopolicity.com/upload/content/pub_1305229189_regular.pdf.

Gido, J. and J. Clements. 2012. Successful Project Management, 5th edition. New York: Cengage Learning.

Hummel, M. L. 2011. “Defeating Maritime Piracy.” Review 6 (2): 157.

Hunt, K. 2012. “Report: Sea Piracy Drops to Lowest Level in Four Years”. CNN.com. http://www.cnn.com/2012/10/23/world/sea-piracy-decline/index.html?hpt=hp_bn2.

ICC-IMB. 2014. “Piracy and Armed Robbery Against Ships: Report for the Period 1 January - 31 December 2013.”

Iida, K. 1992. Studies on the Optimal Search Plan. New York, NY: Springer.

ISO 14825. 2004. Intelligent Transport Systems - Geographic Data Files (GDF). Switzerland: ISO.

Johnson, D. 1977. "Efficient Algorithms for Shortest Paths in Sparse Networks." Journal of the ACM 24. http://dx.doi.org/10.1145%2F321992.321993

Jones, S. 2011. “Security Concerns: Piracy at Sea and the Carriage of Essential Commodities by Merchant Shipping–The Impact on Commodity Pricing and Availability.” Maastricht School of Management Working Papers. ftp://ftp.repec.org/opt/ReDIF/RePEc/msm/wpaper/MSM-WP2011-19.pdf.

Kurose, J. and Ross, K. 2012. Computer Networking: A Top Down Approach, 6th edition. Boston, MA: Pearson.

La Boon, D. 2012. “The Problem of Piracy: The Evolving Military Dynamic.” The Challenge of Piracy Off the Horn of Africa: 107.

Mineau, M. 2010. “Pirates, Blackwater and Maritime Security: The Rise of Private Navies in Response to Modern Piracy.” J. Int’l Bus. & L. 9: 63.

Moody, D. 2006. "40th Anniversary of Palomares." Naval Sea Systems Command Faceplate (10:2): 15-19.

Moreto, W. and J. Caplan. 2010. “Forecasting Global Maritime Policy Using the Risk Terrain Model”. Rutgers Center on Public Security.

Murphy, M. 2009. “Somali Piracy: Not Just a Naval Problem.” Center for Strategic and Budgetary Assessments 16. http://www.csbaonline.org/site/wp-content/uploads/2011/02/2009.04.17-Somali-Piracy-Not-Just-a-Naval-Problem.pdf.

Murphy, M. 2010. Small boats, weak states, dirty money : piracy and maritime terrorism in the modern world. New York: Columbia University Press.

Ng, E. 2014. “World Sea Piracy Drops for Third Straight Year”. Military.com. http://www.military.com/daily-news/2014/01/15/world-sea-piracy-falls-for-third-straight-year.html?ESRC=eb.nl#.Utaqy9U9LGI.mailto

Nincic, D. 2009. “Maritime Piracy in Africa: The Humanitarian Dimension.” African Security Studies 18 (3): 1–16.

PADI. 2003. PADI Search & Recovery Diver Manual. United States: PADI.

Richardson, H. and Discenza, J. 1980. "The United States Coast Guard Computer-Assisted Search Planning System (CASP)." Naval Research Logistics Quarterly, Vol. 27, Number 4. 659-680.

Ritchie, I. 2010. “Life on the Ocean Wave: Prediction Models for Royal Navy Operations”. Meteorology Technology International.

Rosen, K. 2012. Discrete Mathematics and Its Applications, 7th edition. New York, NY. McGraw-Hill.

Scheffler, A. 2010. Piracy-Threat or Nuisance? NATO Defense College, Research Division. http://kms2.isn.ethz.ch/serviceengine/Files/ESDP/113596/ipublicationdocument_singledocument/9dd6ecd2-5d29-4383-92c0-2f03a1c60f92/en/rp_56en.pdf.

Slootmaker, L. 2011. Countering Piracy with the Next-Generation Piracy Performance Surface Model. DTIC Document. http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA543021.

Stone, L. 1975. The Theory of Optimal Search (Mathematics in Science and Engineering). Philadelphia, PA: Elsevier Science.

Stone, L. 1992. "Search for the SS Central America: Mathematical Treasure Hunting." Technical Report of Metron, Inc. Reston, VA.

Stone, L. 2011. "In Search of Air France Flight 447." Institute of Operations Research and the Management Sciences. https://www.informs.org/ORMS-Today/Public-Articles/August-Volume-38-Number-4/In-Search-of-Air-France-Flight-447

Suurballe, J. 1974. "Disjoint Paths in a Network." Networks, 4: 125–145. doi: 10.1002/net.3230040204

Tanenbaum, A. and Wetherall, D. 2010. Computer Networks, 5th edition. Upper Saddle River, NJ: Prentice Hall.

Toth, P. and Vigo, D. 2001. The Vehicle Routing Problem. Philadelphia: Siam.

TSA Carriers. 2015. "Eastbound Bunker Charge Calculator." http://www.tsacarriers.org/calc_bunker.html

United Kingdom Hydrographic Office. 2010. “Anti-Piracy Planning Chart - Red Sea, Gulf of Aden and Arabian Sea.”

U.S. Department of Transportation Maritime Administration. 2011. "Comparison of U.S. and Foreign-Flag Operating Costs."

U.S. Navy. 2015. Photo of Fire Hose Countermeasure. http://www.navy.mil/management/photodb/webphoto/web_090914-N-5345W-246.jpg

U.S. Navy. 2015. Photo of Pirate Mother Ship. http://www.navy.mil/management/photodb/webphoto/web_110202-N-2907P-002.jpg

U.S. Navy. 2015. Photo of Pirate Mother Ship. http://www.navy.mil/management/photodb/webphoto/web_120105-N-ZZ999-001.jpg

U.S. Navy. 2015. Photo of Somali Pirate Skiff. http://www.navy.mil/management/photodb/webphoto/web_090211-N-1082Z-111.jpg

Warbrick, C. 2008. “II. Piracy Off Somalia: UN Security Council Resolution 1816 and IMO Regional Counter-Piracy Efforts.” ICLQ 57: 690–699.

White, R. 2010. "Ocean Shipping Lines Cut Speed to Save Fuel Costs." http://articles.latimes.com/2010/jul/31/business/la-fi-slow-sailing-20100731

Whiteneck, D. 2011. “Piracy Enterprises in Africa.” Center for Naval Analyses Research Memorandum CRM D 23394. http://www.cna.org/sites/default/files/OTA%20Piracy%20Enterprises%20in%20Africa%20D0023394%20A2.pdf.

Winston, W. 2004. Operations Research Applications and Algorithms, 4th edition. Belmont, CA: Brooks/Cole - Thomson Learning.

World Vision Inc. 2014. “Information and Facts About Somalia.” http://www.worldvision.org/our-work/international-work/somalia.