

**OPTIMISING THE PASSAGE THROUGH CHARTED MINEFIELDS BY  
PATH PLANNING AND MINE REMOVAL**

Thesis presented  
in partial fulfilment  
of the requirements for the degree of  
*Master of Science in Industrial Engineering*  
at the University of Stellenbosch



**JÖRG P SCHMID**

**Study Leader: James Bekker**

April 2006

## DECLARATION

I, the undersigned, hereby declare that the work contained in this thesis is my own original work and that I have not previously in its entirety or in part submitted it at any university for a degree.

.....  
6/3/2006  
Date

## ABSTRACT

Shipping is the lifeline to maritime nations. Therefore it is essential that approaches to harbours and other strategic areas are kept free of threats by sea mines. With the technological possibility of remotely surveying threatening sea minefields, it has become necessary to develop a method by which such a charted minefield can be transited with least risk to shipping. To achieve this, two areas of interest have to be addressed and the resulting questions solved. This thesis addresses that requirement by meeting the following objectives:

- to propose a methodology by which the risk involved in transiting a minefield can be managed so that paths of acceptable risks can be taken through a minefield;
- if acceptable paths do not exist, to develop a methodology by which the minimum number of mines can be identified for removal so that a sufficiently safe path is established.

These objectives were met by following the approach outlined below:

- defining the problem in the context of traditional and developing mine warfare and mine countermeasures;
- clearly stating the problems that have to be solved;
- investigating the enablers available to solve the problems;
- selecting and motivating a suitable approach;
- describing the background knowledge to the proposed solutions;
- implementing the solutions in a useable computer application;
- investigating the parameters that pertain to the solution and presenting the findings;
- drawing conclusions from the results and insights obtained from exposure to the problem and the solution strategies.

The presented methodology uniquely combines two methods of combinatorial optimisation to give an integrated solution to the two stated problems of quantifying a risk methodology and removing required mines. The methods use the well known shortest path algorithm, Dijkstra's Algorithm, and a Genetic Algorithm for the basis of the proposed solution. Also, elements of the principle of the Efficient Frontier Graph are integrated to illustrate the aspects of *return versus risk*.

The solution to finding a safe path through a charted minefield is approached from two risk principles:

- Finding a path that is optimised for minimum risk over the entire length of the path. Here risk is a function of the distance between the mine and the ship.
- Defining a maximum allowable risk and minimising the path length. Here risk is translated into the closest distance that a ship is allowed to approach a mine with the areas closer than that, being declared out of bounds.

The sea mine removal problem is solved primarily by using a Genetic Algorithm that bases the quality of a solution on a parameter obtained by applying the methodology developed for solving the problem of optimising a path. This is achieved by minimising the number of sea mines to be removed to create a safe path.



## OPSOMMING

Skeepsvaart is noodsaaklik vir die ekonomiese voortbestaan van maritime nasies. Daarom is dit belangrik dat die bedreiging van seemyne by die toegang tot hawens en ander strategiese seegebiede teëgestaan word. Met die tegnologiese moontlikheid van afstandbeheerde verkenning van bedreigende mynvelde is dit nodig om 'n metode te ontwikkel om so 'n mynveld oor te steek met die minste risiko vir skepe. Om dit moontlik te maak moet twee areas van belang gedek en die gepaardgaande probleme opgelos word. Hierdie tesis dek die vereistes deur die volgende doelstellings te bevredig:

- om 'n metodologie voor te stel waardeur risiko's betrokke by die oorsteek van 'n mynveld bestuur kan word sodat roetes met 'n aanvaarbare risiko deur 'n mynveld gevind kan word;
- indien sulke roetes nie bestaan nie, om 'n metodologie te ontwikkel waarmee die minimum myne vir verwydering geïdentifiseer kan word sodat 'n voldoende veilige roete ontstaan.

Hierdie doelstellings is met die volgende benadering aangespreek:

- deur die probleem in die konteks van tradisionele en ontwikkelende mynoorlogvoering en mynteenmaatreëls te definieer;
- deur die probleme wat opgelos moet word duidelik te stel;
- deur moontlike oplossings tot die probleme te ondersoek;
- deur die keuse en motivering van 'n geskikte benadering;
- deur die agtergrond tot die voorgestelde oplossings te beskryf;
- deur die implementering van die oplossings met 'n rekenaargebaseerde toepassing;
- deur parameters wat verband hou met die oplossing te ondersoek en resultate te toon;
- deur gevolgtrekkings te maak uit die resultate en insigte wat deur blootstelling aan die probleme en oplossingsmetodes verkry is.

Die metodologie wat aangebied word verbind twee metodes vir kombinatoriese optimering op 'n vindingryke manier sodat die twee gedefinieerde probleme van kwantifisering van risiko en verwydering van seemyne aangespreek word. Die metodes maak gebruik van die bekende kortste pad algoritme Dijkstra se Algoritme, en 'n

Genetiese Algoritme as basis vir die voorgestelde oplossing. Dan word daar ook van die beginsels van die “Efficient Frontier Graph” gebruik gemaak om elemente van *opbrengs teen risiko* te illustreer.

Die oplossing om ‘n veilige roete deur ‘n mynveld te vind word vanuit twee risikobeginsels benader:

- Om ‘n roete te vind wat geoptimeer is m.b.t. risiko oor die hele lengte van die roete, met risiko ‘n funksie van afstand tussen die skip en ‘n myn.
- Deur ‘n maksimum toelaatbare lokale risiko te definieer en die roete te optimeer m.b.t. afstand. Hier word risiko vertaal na ‘n minimum toelaatbare afstand tussen myn en skip, sodat enige afstand nader aan ‘n myn as ontoelaatbaar te verklaar.

Mynverwydering is hoofsaaklik opgelos deur ‘n Genetiese Algoritme wat die kwaliteit van ‘n oplossing baseer op ‘n parameter wat deur aanwending van die roete-optimeringsmetode bepaal word. Die algoritme optimeer die verwydering van die minimum aantal myne sodat ‘n veilige pad ontstaan.

## TABLE OF CONTENTS

<b>CHAPTER 1</b>	<b>INTRODUCTION</b>	<b>1</b>
1.1	Context and importance of this thesis	1
1.2	A short history of mine warfare	2
1.2.1	<i>The sea mine</i>	2
1.2.2	<i>Mine countermeasures</i>	3
1.3	Structure of the thesis	5
1.4	Problem formulation	6
<b>CHAPTER 2</b>	<b>SAFE PATH OPTIMISATION</b>	<b>7</b>
2.1	Alternative approaches	7
2.2	Proposed solution	9
2.2.1	<i>Motivation</i>	10
2.2.2	<i>Simplifications and assumptions</i>	10
2.2.3	<i>Graph theory</i>	11
2.2.4	<i>Dijkstra's Algorithm</i>	12
2.2.5	<i>Efficient Frontier Graph</i>	19
2.2.6	<i>General description of the solution implementation</i>	21
2.2.7	<i>Quantification of arc cost function in terms of risk</i>	23
2.2.8	<i>Quantification of path risk</i>	29
2.2.9	<i>Quantification of path length</i>	29
2.2.10	<i>Introduction of safety radius</i>	30
2.3	Computer implementation	30
2.4	Parameters of importance to the implementation	33
2.4.1	<i>Grid spacing</i>	33
2.4.2	<i>Arc risk cost function</i>	33
2.4.3	<i>Safety radius</i>	34
2.4.4	<i>Risk weighting</i>	34
2.5	Sample results for path optimisation	34
2.5.1	<i>Grid spacing for a specific minefield</i>	35
2.5.2	<i>Selection of a suitable grid spacing</i>	40
2.5.3	<i>Grid spacing selection in general terms</i>	43
2.5.4	<i>Paths for ArcRisk = f(closest mine) and ArcRisk = f(all mines)</i>	48
2.5.5	<i>Paths for different k values</i>	49

2.5.6	<i>Single and multiple start nodes and end nodes</i> .....	50
2.5.7	<i>Safest paths with and without safety radius (specific case)</i> .....	51
2.5.8	<i>Safest paths with and without safety radius (general trends)</i> .....	52
2.5.9	<i>Shortest paths with changing safety radius</i> .....	54
2.5.10	<i>Risk weighting setting</i> .....	57
2.6	Conclusions on path optimisation investigation .....	60
<b>CHAPTER 3 MINE REMOVAL OPTIMISATION</b> .....		62
3.1	The Knapsack Problem .....	63
3.2	Genetic Algorithms .....	64
3.3	Genetic Algorithm solution implementation .....	65
3.4	Random search algorithm .....	68
3.5	Results of mine removal optimisation implementation .....	70
3.5.1	<i>Investigation of crossover and population selection criteria</i> .....	70
3.5.2	<i>Changing the safety distance</i> .....	71
3.5.3	<i>Results of changing the population size</i> .....	74
3.5.4	<i>Results of the random search algorithm</i> .....	75
3.6	Conclusion for mine removal optimisation .....	76
<b>CHAPTER 4 SUMMARY, CONCLUSIONS AND RECOMMENDATIONS</b> .....		77
4.1	Summary of the research .....	77
4.2	Conclusions on the research .....	77
4.2.1	<i>Path optimisation</i> .....	79
4.2.2	<i>Mine removal</i> .....	79
4.3	Recommendations for further work .....	80
4.3.1	<i>General approach</i> .....	80
4.3.2	<i>Dijkstra's Algorithm for path finding</i> .....	81
4.3.3	<i>Mine removal optimisation</i> .....	81
<b>REFERENCES</b> .....		82
<b>APPENDIX A: COMPUTER LISTING OF ROUTEOPTIMISER BLOCK</b> .....		A-1
<b>APPENDIX B: COMPUTER LISTING OF GAMINEOPTIMISER BLOCK</b> .....		B-1

## LIST OF FIGURES

Figure 2-1: Simplified method of finding the shortest sufficiently safe path .....	7
Figure 2-2: Voronoi diagrams of random minefields represented in (a) three dimensions and (b) two dimensions .....	8
Figure 2-3: Terms and conventions used in graph theory .....	12
Figure 2-4: Pseudo code of Dijkstra's algorithm .....	14
Figure 2-5: The area represented by different portfolios' return <i>versus</i> risk for a specified period ([16]) .....	20
Figure 2-6: Example of an efficient frontier graph ([17]) .....	21
Figure 2-7: The routing structure and some nomenclature .....	22
Figure 2-8: The concept of mines with safety radius .....	23
Figure 2-9: Assigning a risk to an arc as a function of the distance to the closest mine ..	27
Figure 2-10: Simplified flow diagram for Route Optimiser block .....	31
Figure 2-11: Extend <sup>®</sup> model elements and reduced user interface for path optimisation ..	32
Figure 2-12: Route Optimiser block detail and important user interface .....	32
Figure 2-13: Layout of operational area and minefield for the displayed results .....	35
Figure 2-14: Paths generated using different grid spacing (detail in Table 2-3) .....	37
Figure 2-15: Path risk <i>versus</i> grid spacing .....	39
Figure 2-16: The effect of grid spacing on output accuracy and solution time .....	40
Figure 2-17: <i>Return versus Risk</i> resulting from changing grid resolution .....	41
Figure 2-18: Modified <i>Return versus Risk</i> graph for grid resolution .....	42
Figure 2-19: Path risk confidence interval for 20 randomly generated minefields as a function of grid spacing .....	45
Figure 2-20: Path risk accuracy confidence interval of 20 randomly generated minefields as a function of grid spacing .....	47
Figure 2-21: Path length accuracy confidence interval for of 20 randomly generated minefields with 200m safety radius as a function of grid spacing .....	48
Figure 2-22: Safest paths generated for arc risk as (a) a function of the closest mine and (b) all mines .....	49
Figure 2-23: Paths generated for different <i>k</i> values .....	50
Figure 2-24: Results of path length and risk improvement by allowing multiple start and end nodes .....	51
Figure 2-25: Path change with safety radius increase from (a) 200m to (b) 220m .....	52

Figure 2-26: Change of path risk and path length with increasing safety radius.....	55
Figure 2-27: Return <i>versus</i> risk for increasing safety radius .....	56
Figure 2-28: Paths for (a) 0% and (b) 9% risk weighting setting .....	57
Figure 2-29: Path risk and path length <i>versus</i> risk weighting.....	58
Figure 2-30: Efficient Frontier Graph for the selected minefield, generated by varying the risk weighting .....	59
Figure 2-31: Enlarged area from Figure 2-30, showing Efficient Frontier .....	60
Figure 3-1: Pseudo code of a Genetic Algorithm .....	65
Figure 3-2: Extend <sup>®</sup> genetic algorithm optimisation model with reduced user interface..	69
Figure 3-3: Model detail and user interface relating to genetic algorithm parameters .....	69
Figure 3-4: Safe path resulting from 8 mines being removed, following a genetic algorithm optimisation (200m safety radius).....	72
Figure 3-5: Convergence of the genetic algorithm with fitness of the best chromosome and number of mines removed per generation.....	73
Figure 3-6: New path for a smaller safety radius (150m) requiring removal of 2 mines .	73
Figure 3-7: New path for a larger safety radius (250m) requiring removal of 14 mines...	74

**LIST OF TABLES**

Table 2-1: Graphic representation and data structures associated with a simple example of Dijkstra's Algorithm applied to a network .....	15
Table 2-2: Example of modification to Dijkstra's Algorithm to allow multiple end nodes .....	18
Table 2-3: Summary of effect of grid spacing on optimisation output.....	38
Table 2-4: Result accuracy using $0.0269\text{m}^{-1}$ as path risk reference .....	39
Table 2-5: Path risk values (in $\text{m}^{-1}$ ) and their analysis for 20 randomly generated minefields.....	44
Table 2-6: Path risk accuracy values (in %) and their analysis for 20 randomly generated minefields vs grid spacing.....	46
Table 2-7: Magnitude of safety radius in order to affect path optimisation .....	53
Table 3-1: Results of genetic algorithm performance as a function of replication parameters (for 20 repetitions).....	71
Table 3-2: Results of GA implemented with different population sizes (for 20 repetitions) .....	75
Table 3-3: Results of Random Search Algorithm performance with and without genetic algorithm implementation (for 20 repetitions).....	76

## CHAPTER 1 INTRODUCTION

### 1.1 Context and importance of this thesis

The methodology and processes described in this thesis originated from a requirement to facilitate safe routing through a charted minefield. Elements of the thesis can be equally applied to other routing problems, particularly those where no distinct paths exist. The application for safe routing as described here has its significance in modern sea mine countermeasures, the importance of which has to be explained against the history of sea mine warfare and the methods developed over time to combat the sea mine threat. The term *mine* has lately become notorious because it is primarily associated with *landmines* which indiscriminately maim people. In contrast, *sea mines* have escaped much of this stigma and are still a strategic tool in the arsenal of anyone with military intentions at sea<sup>1</sup>. The greatest short-term modern threat comes from their use as a way to impede merchant sea traffic by closing approaches to harbours or important sea passages. Most sea-going nations depend heavily on maintaining active harbours, supporting both the bulk of their import and export requirements to sustain the economy. It is estimated that South Africa's seaborne trade is 95% by volume and 80% by value. This makes the Republic of South Africa one of the 12 largest maritime trading nations of the world and highlights the economic importance of its harbours [1]. This trade is made up of both bulk raw materials and produce that are exported and processed goods that are imported and exported. For economic survival of this country, it is therefore vital that its harbours are kept open at all times. In the event that the harbours should be blocked, it is essential that they be opened to normal shipping as soon as possible. With the event of asymmetric warfare (small groups of ill-equipped fighters attacking powerful nations) and international terrorism, the threat of harbours being blocked by sea mines cannot be ignored and provision has to be made to minimise this risk.

Additionally, the use of sea mines is an important method for preventing naval operations from maintaining free movement at sea. Any navy that wishes to defend itself against an

---

<sup>1</sup> For the sake of brevity and due to the context within which this study is performed, the geographical location (i.e. sea as opposed to land) of a mine will be omitted. Instead the term mine will refer to a sea mine unless otherwise stated.



aggressor or fight at sea needs this movement to remain unimpeded. Mines are readily used for locking enemy ships into their home harbours or preventing them from operating freely in areas of strategic or tactical importance.

## 1.2 A short history of mine warfare

The term *mine warfare* encompasses the application of sea mines and measures taken to counter them. It is important to note that the development of both these areas was interdependent, with better mines requiring better countermeasures, which in turn resulted in a requirement for improved mines. This played off against the growth in technological enablers that pushed the development of both the mine and its countermeasures. A good historical overview is provided in [2] and [3]. Reference [2] also includes an extended section on the Hague Convention of 1907 which attempted to establish the rules and restriction of using sea mines. This can be indirectly viewed as an international response to minimise the economic threat that the uncontrolled use of sea mines would pose to the global community.

### 1.2.1 The sea mine

Sea mines developed from crude gunpowder-filled kegs into sophisticated micro-processor controlled weapons in about 200 years. Early mines were designed to be detonated via electrical link to shore and later developed into weapons that would detonate on contact with a ship. Variants of these contact mines still exist and are still used in conflicts at sea. But a more recent development is the influence mine which detonates when certain requirements are met when its sensors sense the presence of a ship-like target and its internal logic or artificial intelligence fulfil preset limits. The most typical ship influences used for this are:

- acoustic signatures – generated mainly by machinery and propellers;
- magnetic signatures – present because they are inherent to the ship's construction or induced by the earth's magnetic field;
- pressure signatures – a result of the pressure fluctuations created by the displacement of water around the ship's hull.

Other signatures are also used in more sophisticated mines and can include for example electric fields. These mines are most commonly designed to rest on the sea bottom (bottom mines) or moored below the sea surface (moored influence mines).

The combination of the ship's characteristic signatures (e.g. modulation, frequency components, amplitude) and the mine's capabilities (e.g. sensor type, sensitivity, logic) determine the likelihood that a mine will trigger in a position to cause damage to a ship.

### *1.2.2 Mine countermeasures*

The field of mine countermeasures (MCM) has developed in parallel to the sea mine and almost all of its evolution has been in direct response to advances in mine technology. Consequently the countermeasures have always lagged behind the mine capabilities by some distance. Even with the technological gap at its closest, it has always been extraordinarily more difficult to counter a mine threat than to create it. Initial attempts were based mainly on preventing mines from being laid or minefields from being deployed. Soon the process of mechanical minesweeping was developed, where moored mines were mechanically separated from their anchors, bringing them to the surface of the sea where they could be disposed of, mostly through gunfire.

The next step was in response to influence mines where influence minesweeping was used. Here an attempt was made to trigger the mine by stimulating its sensors artificially so that the mine would detonate. This was achieved by towing gear that simulated acoustic and magnetic signatures behind a ship. The mines would detonate at a safe distance behind the ship. This process still contained risk, both from the uncertainty whether all mines had been destroyed or not and from the fact that the towing ship had to travel over active mines.

With advances in sonar technology, systems could be created with which it was possible to detect mines on the sea bed and moored mines in the water column at a sufficient distance ahead of the ship carrying the sonar. Once detected, the mines could be neutralised by divers or remotely operated vehicles (ROVs) specially designed for this purpose, by releasing a charge next to the mine that would be detonated from a safe distance. These methods were seen as a major breakthrough and were thought to be sufficient to take over the bulk of mine neutralisation tasks from minesweeping. Drawbacks were their slow speed of advance and the fact that in certain conditions (e.g. reef bottom, unsuitable sonar conditions) the detection probability was insufficient. In addition smaller and stealthier mines were being developed and deployed, further reducing the chance of finding all mines and increasing the risk to the vessels.

The next and latest major development in mine countermeasures was the move to remotely operated or autonomous systems. Here the principle is twofold, firstly removing personnel from the vicinity of the mines and secondly bringing the sonar to a position where detection probability is increased by operating more closely to the sea bottom and reducing the distance between the sonar and its target. It has now become possible to map large areas of potential minefields with a very high positional accuracy, probability of detection and correct classification. Since it is now possible to obtain such information remotely, it can be used to plan a path through the minefield for ships that have to transit the minefield. This can make the process of breaching the minefield much quicker, as only a selected subset of the mines in the minefield require removal, if any at all. This method of mine countermeasures is sometimes called mine avoidance. The term *mine avoidance* is also used for the in-stride process where a manned ship equipped with an active mine-detecting sonar ensures it maintains a safe distance from all detected mines while moving through a minefield.

In contrast to this, the proposed methodology described in this thesis deals more specifically with the scenario where a minefield or selected area is fully charted, after which the resulting information is used for path optimisation. The distinct disadvantage of this methodology is that a path can generally not be planned and attempted until the complete mapping results are considered as a whole. This differs very much from the conventional mine-hunting process where neutralisation immediately follows detection and classification of a mine. It is, however, very well adaptable to a scenario where the mapping is done without it being evident to the enemy, and the clearance and breaching can then be achieved much more speedily than conventionally. It is also applicable to the scenario where submarines have to transit minefields undetected.

The above factors create the background for the subject matter with which this thesis deals. The methodology with which this topic is approached is expanded in the following chapters.

### 1.3 Structure of the thesis

This thesis contains the elements normally expected in a technical investigation. The progression with which the subject matter is approached may however not be entirely conventional. This chapter aims to provide the overview of how the topics have been structured to introduce the logical progression of solution development.

The background that is required to make the reader familiar with the context of the thesis was presented in §1.1- §1.2. Once the broad context has been sketched there, in § 1.3 the reader will be guided into the framework of how the subject matter is dealt with. In § 1.4 the questions posed in the introductory discussion will be formalised. Thereafter the problem solution will be addressed in CHAPTER 2 and CHAPTER 3, which together form the main part of the technical description of the thesis. Finally the work is summarised and conclusions drawn in CHAPTER 4 together with possible recommendations for future avenues of research.

The literature survey that relate to the specialist areas required for the problem resolution are addressed in context to the topics. For that reason a separate literature survey is not included other than the short introduction to mine warfare in § 1.1 - § 1.2. Instead the theoretical background is addressed in each chapter as required, forming the starting point for the solution implementation. The detail of the two solution chapters is:

- Content: CHAPTER 2 deals with the proposed solution to the problem of finding a safe path through a charted minefield without mine removal.  
CHAPTER 3 describes the methods with which the least and fewest number of mines can be selected for removal so that safe transit is possible.
- Structure: Both chapters follow the same logical structure. They consist of a discussion of the alternative approaches to solve the relevant problem. The selected solution is described in theoretical terms (including the contextual literature requirements) before the computer implementation is presented. Thereafter the important aspects of the solution that require consideration are addressed by presentation and discussion of sample results.

## 1.4 Problem formulation

The problem that has to be solved is that of creating a path<sup>2</sup> through a charted minefield in such a way that a path of acceptable risk to the transiting ship results. The level of risk that is deemed acceptable can vary according to circumstances. In important military operations it may be required to minimise the time that the breaching operation takes and a higher risk will be accepted. In other instances, for example if the vessel transiting the minefield is of high value, it may be imperative to generate the safest path regardless of the resulting path length. This implies a trade-off between path length (transiting time) and path risk.

To solve the problem of finding a sufficiently safe path through a minefield, two distinct processes are required. The **first**, which is always required, is a methodology with which factors affecting risk can be quantified and the **path of acceptable risk can be selected**. This path can either be:

- on the one extreme, the *shortest* path within acceptable risk parameters;
- on the other extreme the *safest* path according to certain selected risk parameters; or
- a selected set value between the two extremes, based on risk parameter preferences set by the decision maker.

The **second** part to the problem is applicable when a sufficiently safe path cannot be found by the previous method. It now becomes necessary **to identify an optimum number of mines which can be removed** in order for a safe path to result.

Both these methodologies require optimisation of a combinatorial type. In the first instance the correct sequence of path sections has to be found so that an optimal path results through the minefield. The second requires the selection of the best combination of mines that have to be removed in order to create a sufficiently safe path. In this thesis the two problem parts are addressed separately although, they are interdependent, as will become evident later.

---

<sup>2</sup> In this study the term *path* has been used in preference to *route*. *Path finding* is an accepted study field in graph theory e.g. [5], while *route finding* is used more outside the scientific domain.



## CHAPTER 2 SAFE PATH OPTIMISATION

In this chapter the methodology with which to solve the first part of the stated problem of finding a sufficiently safe path through a minefield is presented. Alternative approaches that can be followed are presented in 2.1, before the selected solution approach used for this study is motivated in § 2.2. The theoretical background required for the solution is given, whereafter the implementation is discussed, both in creation of a software-based tool and in the presentation of sample results.

### 2.1 Alternative approaches

Intuitive reasoning related to the problem leads to two solutions to the routing problem, one providing the shortest path and the other the safest path. The shortest path can be found by defining an area around each mine where it is too dangerous to move through. This area is called the *safety area* and as it is assumed to be a circle, it may be called a *safety circle*. It is defined by its radius, which may be described as the *safety radius*. The concept of its quantification is described in § 2.2.7 (p. 23). Areas where safety circles do not meet or intersect are considered passable. The shortest path is taken through passable areas when the safety circle is skirted, resulting in straight line segments joined by arcs which form part of the circle generated by the safety radius of a mine. This concept is represented diagrammatically in Figure 2-1.

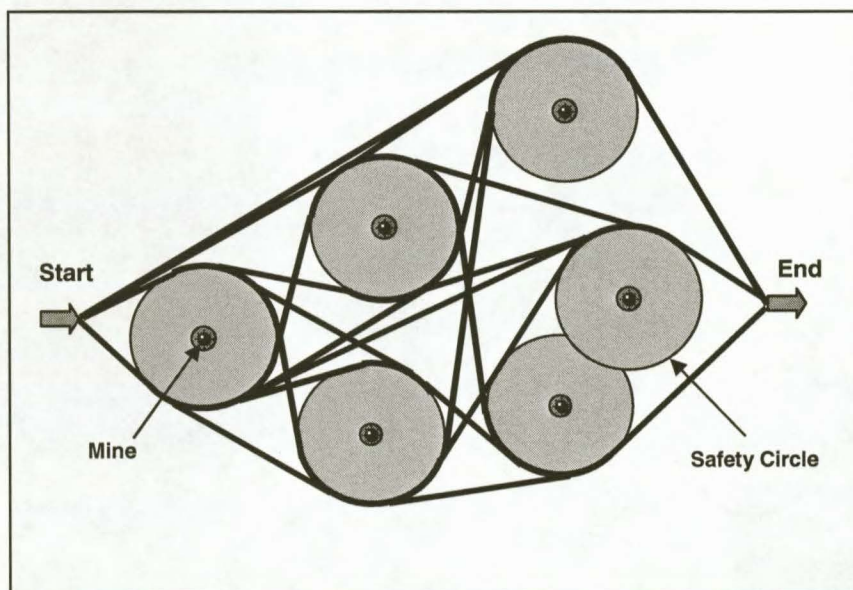
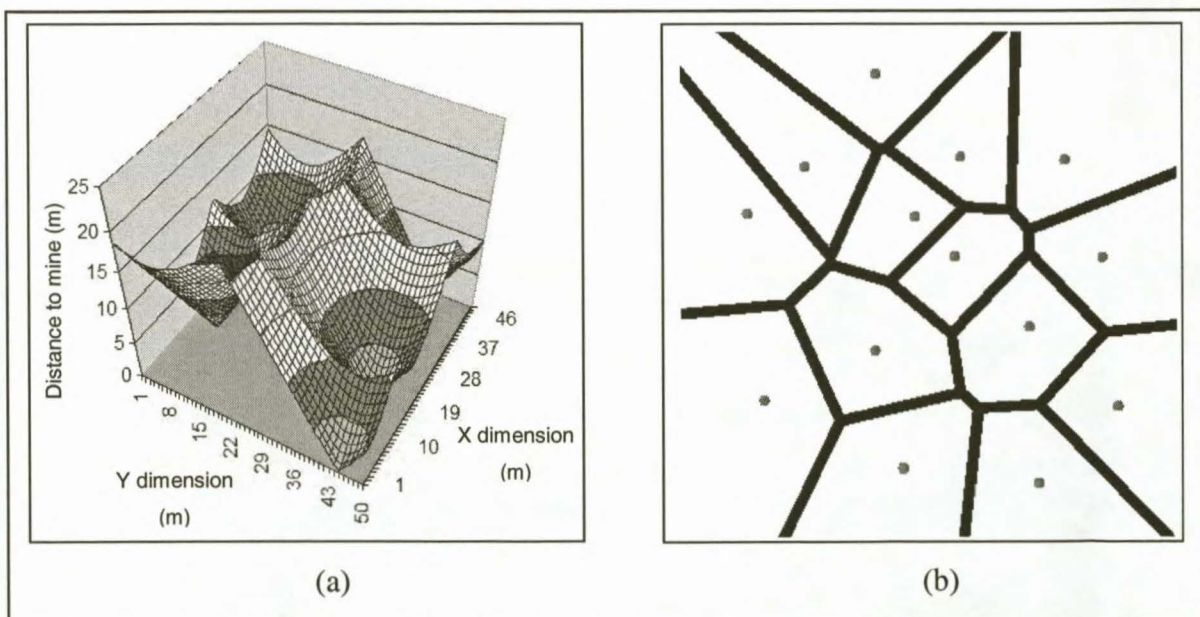


Figure 2-1: Simplified method of finding the shortest sufficiently safe path



Different possible solutions are shown, still requiring the shortest path to be identified by selection of the correct combination of path segments between start- and end-points. The smaller the safety radius, the more direct the shortest path will be, but the closer a ship will have to pass the mines and the more dangerous the path becomes.

The safest path is generated if equal distance is kept to two mines when passing between them. As the minefield can only be transited by passing between mines, the safest path is that path that spends most of its time at the maximum possible distance from all mines it has to pass. A network of possible paths is generated by a Voronoi diagram [4] and is shown in Figure 2-2 (a) as a three-dimensional plot with the third dimension representing the distance to the closest mine. When viewed from the top, the ridges representing equal distances to the two closest mines appear as straight lines, provided the mine is regarded as a point as shown in Figure 2-2 (b). Note that the two figures do not represent the same combination of mines. Optimisation is now required to select the sequence of arcs representing the safest path from the required start point to the end points of the transit. The safest path would be that path which has the least cumulative risk, where risk for a unit length of the path can be expressed as a function of the vertical dimension (distance to the closest mines) in the Figure 2-2 (a).



**Figure 2-2: Voronoi diagrams of random minefields represented in (a) three dimensions and (b) two dimensions**



The two simplified methods described do not provide sufficient freedom in manipulating the risk value for selected paths. Their shortcomings will become clearer when viewed against the proposed method in § 2.2. They have the advantage of simplicity (although the generation of a Voronoi diagram is numerically complex) and would be able to provide quick answers due to their hugely reduced complexity (number of possible combinations) when viewed against the proposed method. However, with modern computing power available, this is not seen as sufficient motivation to implement them over the proposed method.

The parallels to networks requiring solutions of the shortest path are evident and were the starting point of a solution to the safe routing problem. The theory and alternatives dealing with such optimisation are described in detail in [5].

Literature searches into practical applications initially led to two related applications, both using similar methods of finding best paths. The first and simplest was related to gaming applications where paths are often required for gaming entities to move from one position in the game to another. Further searches provided examples in the military domain, robot navigation, terrain navigation and others ([6], [7], [8]), which can be extrapolated to minefield routing.

Additional research led to a proposed solution for finding a safe path through a naval minefield using integer programming techniques and a commercial solver [9]. Some of the principles developed by that and other authors have been included in this thesis.

## **2.2 Proposed solution**

In this paragraph the selected methodology is discussed before being implemented in a computer application in § 2.3. The theoretical principles required during the implementation are presented together with the motivation for their selection and simplifications or assumptions to make them valid for use here. *Graph theory* is an important foundation for the proposed solution and therefore it is shortly described. The topic of the *Efficient Frontier* is also discussed, although it is not directly relevant to the computer implementation of the optimisation technique. It does, however, relate to the physical implementation of the



methodology in finding solutions to specific operational problems where parameters of the methodology have to be evaluated and selected for use.

### 2.2.1 *Motivation*

The solution that was sought, required a user-adjustable variation that could trade off *path risk* and *path length*. This would have to enable infinitely variable solutions lying between the two extremes. As infinitely variable problems provide a solution space too large to solve, it was required to generate a grid-based solution providing a finite, albeit large, number of possible path combinations. This method, as well as those simplified approaches shown above, provides a network of possible paths, requiring an optimisation methodology to find a path of the suitable risk-profile.

There was the additional requirement (not described in this thesis) to expand the routing optimisation to include threats that were spatially not simple to describe. This was implemented to include distributed risks such as environmental factors (for example sea bottom types) and was dependent on a solution where the spatial distribution could easily be translated into the routing solution to create risk constraints.

### 2.2.2 *Simplifications and assumptions*

The problem of a ship encountering a mine is a three-dimensional occurrence with the important parameters of depth of the mine under the sea surface and the draught of the ship. For the purpose of this thesis the problem has been reduced to two dimensions, a X-Y plane aligned with the idealised two-dimensional surface of the sea. As the important parameter of distance between mine and ship is used extensively in this thesis, it has for that purpose been reduced to the two-dimensional orthogonal distance in the plane. Depth data would normally be available from a high accuracy sea-bottom survey, and in real-life situations the distance parameter should include the slant range so that the true three-dimensional distance is used in determining risk. The implementation of this is easily achieved and its exclusion for the purpose described hereof has no bearing on the principles of optimisation.

An additional simplification is that the ship orientation to the mine (*i.e.* its heading and how large the lateral component is in the three-dimensional distance vector) is completely ignored in the solution described, with the ship being considered as a point in the plane. This may

influence the effect of a detonating mine on a ship. If required, it could be introduced as additional parameters in the risk equation for real-life implementation.

### 2.2.3 Graph theory

As will become evident below, the proposed solution includes elements of graph theory. For that reason a short theoretical overview of the applicable theoretical background is given here, obtained primarily from [5] and [10]. A specific solution based on the principles of graph theory is described in this chapter. However the applications of graph theory stretch over many fields, including the related field of transportation planning and more diverse areas such as chemistry, industrial and electronic engineering, management, project planning and others. As such it has evolved into a specialist branch in mathematics.

The generally accepted starting point for graph theory was the paper by Swiss mathematician Leonard Euler published in 1736. In it he described a routing problem (the modern equivalent is known as the *postman problem*) concerning an island in a river connected by a number of bridges to the mainland. Since then, two distinct directions have developed as *algebraic* or theoretical graph theory on the one hand and graph theory dealing with *optimisation* techniques. The optimisation field has greatly benefited from computer developments, which have made solutions such as the one described here, possible.

A graph consists of a set of elements called *vertices* which are connected to each other by links called *edges*. This can be formulated in the following way:

An **undirected graph** or **graph**  $G$  is an ordered pair  $G:=(V,E)$  with

- $V$ , a set of *vertices* or *nodes*
- $E$ , a set of unordered pairs of distinct vertices, called **edges**. The vertices belonging to an edge are called the **ends**, **endpoints** or **end vertices** of the edge.
- $V$  and hence  $E$  are usually taken to be finite sets.

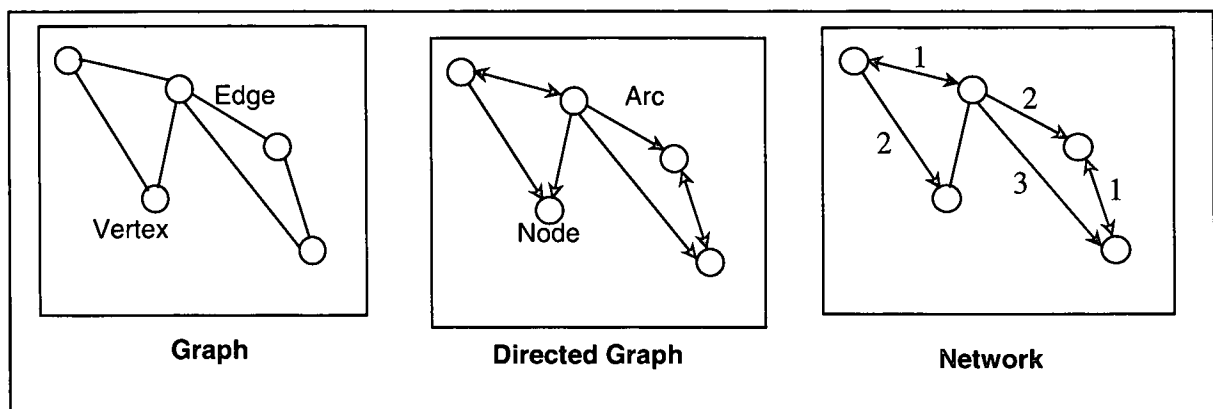
A **directed graph** or **digraph**  $G$  is an ordered pair  $G:=(V,A)$  with

- $V$ , a finite set of *vertices* or *nodes*

- A, a finite set of ordered pairs of distinct vertices, called **directed edges, arcs** or **arrows**. An arc  $a=(x,y)$  is considered directed from  $x$  to  $y$  with  $y$  called the head and  $x$  called the tail.

The terms *nodes* and *arcs* are used in this thesis and refer to the graph elements and the links between them.

A *network* is a graph that has values (one or more) associated with the arcs. These values can typically be length, cost, time, reliability or other relevant parameters depending on the application. The optimisation of graphs and networks forms an important area of operations research. The terminology is graphically explained in Figure 2-3.



**Figure 2-3: Terms and conventions used in graph theory**

#### 2.2.4 *Dijkstra's Algorithm*

Dijkstra's Algorithm<sup>3</sup> is a generally accepted efficient method to solve for shortest paths in a network. It can be used to determine the shortest path between a starting node and all other nodes in a network (one-to-many). In a modified form it is useable to determine the shortest distance between a chosen starting and end node (one-to-one). It can also be adapted to provide the shortest distance between all nodes (many-to-many), but more effective algorithms exist to achieve this.

<sup>3</sup> Also referred to as Dijkstra's single source algorithm

The algorithm is based on a systematic procedure whereby adjacent nodes are examined, beginning at the start or end node and updated sequentially each time a shorter path is found. It is easily implemented in a computer language and gives a quick, exact solution to the one-to-one or one-to-many problem. For the case of a one-to-one solution, the algorithm can be improved by implementation of the A\* algorithm which includes a heuristic element in the selection of a next node to be investigated. The A\* algorithm chooses as the next node not the one with the shortest distance to the beginning node as Dijkstra's algorithm does. Instead, the node with the smallest sum of the distance to the beginning node and the straight line distance to the end node is chosen as the next node. This forces the paths leading more directly to the end node to be given priority, making the solution somewhat faster in most cases, especially where the path between the start and end node is reasonably direct. Further information on the A\* algorithm can be found in [11].

Dijkstra's algorithm is executed as follows: Let  $V$  be the set of  $n$  nodes in a directed graph, and with each node  $v_i \in V$  we associate an auxiliary cost variable  $d_i \in D$ . Let  $cost(v_j, v_k)$  denote the cost (distance or risk) to move from node  $v_j$  to node  $v_k$ . Initially, let  $T = V$ ,  $d_1 = 0$  and  $d_i = \infty$ , for  $2 \leq i \leq n$ . During each iteration the node  $v_i \in T$  with the smallest value of  $d_i$  is removed. Each neighbour  $v_m$  of  $v_i$  is updated if the associated cost  $d_m$  is less than the existing cost. This is done until all nodes have been removed from  $T$ . When a node is removed from  $T$ , it is inserted into the set  $S$ , which is initially empty. When the algorithm terminates,  $S$  contains the shortest paths in the graph from  $v_i \in V$  to all  $v_j \in V$  ( $i \neq j$ ). In pseudo code, the algorithm is shown in Figure 2-4.

```
Procedure Dijkstra
Begin
  For Each element in  $V$ 
     $d_i \leftarrow \infty$ 
  End For
   $d_1 \leftarrow 0$ 
   $T \leftarrow V$ 
   $S \leftarrow \emptyset$ 
  Do While  $T$  is not an empty set
     $i \leftarrow$  Index of node in  $T$  of which  $d_i$  is minimum
    For Each arc  $(v_u, v_m)$  outgoing from  $v_u$ 
      If  $d_m > (d_u + \text{cost}(v_u, v_m))$  Then
         $d_m \leftarrow d_u + \text{cost}(v_u, v_m)$ 
      End If
       $T \leftarrow T - v_m$ 
       $S \leftarrow S + v_m$ 
    End For
  Do Loop
End
```

Figure 2-4: Pseudo code of Dijkstra's algorithm

The algorithm is demonstrated below (Table 2-1) on a simple graph for the case of one-to-all, showing both a graphic representation and the associated data structures that would be required for computer implementation, adapted from [12] and [13].

**Table 2-1: Graphic representation and data structures associated with a simple example of Dijkstra's Algorithm applied to a network**

Network	Actions	Data Structure																								
	<p>Initialise all labels to <math>\infty</math> except the starting node <math>a</math> to 0.</p>	<table border="1"> <thead> <tr> <th><math>a</math></th> <th><math>b</math></th> <th><math>c</math></th> <th><math>d</math></th> <th><math>e</math></th> <th>Nodes</th> </tr> </thead> <tbody> <tr> <td></td> <td></td> <td></td> <td></td> <td></td> <td>Compl.</td> </tr> <tr> <td>0</td> <td><math>\infty</math></td> <td><math>\infty</math></td> <td><math>\infty</math></td> <td><math>\infty</math></td> <td>Label</td> </tr> <tr> <td>-</td> <td>-</td> <td>-</td> <td>-</td> <td>-</td> <td>Pred.</td> </tr> </tbody> </table>	$a$	$b$	$c$	$d$	$e$	Nodes						Compl.	0	$\infty$	$\infty$	$\infty$	$\infty$	Label	-	-	-	-	-	Pred.
$a$	$b$	$c$	$d$	$e$	Nodes																					
					Compl.																					
0	$\infty$	$\infty$	$\infty$	$\infty$	Label																					
-	-	-	-	-	Pred.																					
	<p>Relax all nodes adjacent (bold arrows) to <math>a</math> by updating their distance to the starting node <math>a</math>.</p> <p>Update the relaxed nodes with their predecessor (in this case node <math>a</math>).</p> <p>Mark node <math>a</math> as completed.</p>	<table border="1"> <thead> <tr> <th><math>a</math></th> <th><math>b</math></th> <th><math>c</math></th> <th><math>d</math></th> <th><math>e</math></th> <th>Nodes</th> </tr> </thead> <tbody> <tr> <td>*</td> <td></td> <td></td> <td></td> <td></td> <td>Compl.</td> </tr> <tr> <td>0</td> <td>10</td> <td>5</td> <td><math>\infty</math></td> <td><math>\infty</math></td> <td>Label</td> </tr> <tr> <td>-</td> <td><math>a</math></td> <td><math>a</math></td> <td>-</td> <td>-</td> <td>Pred.</td> </tr> </tbody> </table>	$a$	$b$	$c$	$d$	$e$	Nodes	*					Compl.	0	10	5	$\infty$	$\infty$	Label	-	$a$	$a$	-	-	Pred.
$a$	$b$	$c$	$d$	$e$	Nodes																					
*					Compl.																					
0	10	5	$\infty$	$\infty$	Label																					
-	$a$	$a$	-	-	Pred.																					
	<p>Select next closest node to the starting node (node <math>c</math>) and relax all adjacent nodes (bold arrows).</p> <p>If less than the present distance label, update the distance label of the node and the predecessor.</p> <p>Mark node <math>c</math> as completed.</p>	<table border="1"> <thead> <tr> <th><math>a</math></th> <th><math>b</math></th> <th><math>c</math></th> <th><math>d</math></th> <th><math>e</math></th> <th>Nodes</th> </tr> </thead> <tbody> <tr> <td>*</td> <td></td> <td>*</td> <td></td> <td></td> <td>Compl.</td> </tr> <tr> <td>0</td> <td>8</td> <td>5</td> <td>14</td> <td>7</td> <td>Label</td> </tr> <tr> <td>-</td> <td><math>c</math></td> <td><math>a</math></td> <td><math>c</math></td> <td><math>c</math></td> <td>Pred.</td> </tr> </tbody> </table>	$a$	$b$	$c$	$d$	$e$	Nodes	*		*			Compl.	0	8	5	14	7	Label	-	$c$	$a$	$c$	$c$	Pred.
$a$	$b$	$c$	$d$	$e$	Nodes																					
*		*			Compl.																					
0	8	5	14	7	Label																					
-	$c$	$a$	$c$	$c$	Pred.																					
	<p>Select next closest node to the starting node (node <math>e</math>) and relax all adjacent nodes (bold arrow).</p> <p>If less than the present distance label, update the distance label of the node and the predecessor.</p> <p>Mark node <math>e</math> as completed.</p>	<table border="1"> <thead> <tr> <th><math>a</math></th> <th><math>b</math></th> <th><math>c</math></th> <th><math>d</math></th> <th><math>e</math></th> <th>Nodes</th> </tr> </thead> <tbody> <tr> <td>*</td> <td></td> <td>*</td> <td></td> <td>*</td> <td>Compl.</td> </tr> <tr> <td>0</td> <td>8</td> <td>5</td> <td>13</td> <td>7</td> <td>Label</td> </tr> <tr> <td>-</td> <td><math>c</math></td> <td><math>a</math></td> <td><math>e</math></td> <td><math>c</math></td> <td>Pred.</td> </tr> </tbody> </table>	$a$	$b$	$c$	$d$	$e$	Nodes	*		*		*	Compl.	0	8	5	13	7	Label	-	$c$	$a$	$e$	$c$	Pred.
$a$	$b$	$c$	$d$	$e$	Nodes																					
*		*		*	Compl.																					
0	8	5	13	7	Label																					
-	$c$	$a$	$e$	$c$	Pred.																					



Table 2-1 continued

<p>The graph shows nodes a, b, c, d, e. Node a has a distance of 0. Node b has a distance of 8. Node c has a distance of 5. Node d has a distance of 9. Node e has a distance of 7. Edges and their weights: a to b (10), a to c (5), b to d (1), b to c (2), c to b (3), c to d (9), c to e (2), d to c (4), e to d (6). Bold arrows indicate relaxation: a to b, a to c, b to d, c to b, c to e, d to c, e to d.</p>	<p>Select next closest node to the starting node (node <i>b</i>) and relax all adjacent nodes (bold arrows).</p> <p>If less than the present distance label, update the distance label of the node and the predecessor.</p> <p>Mark node <i>b</i> as completed.</p>	<table border="1"> <thead> <tr> <th><i>a</i></th> <th><i>b</i></th> <th><i>c</i></th> <th><i>d</i></th> <th><i>e</i></th> <th>Nodes</th> </tr> </thead> <tbody> <tr> <td>*</td> <td>*</td> <td>*</td> <td></td> <td>*</td> <td>Compl.</td> </tr> <tr> <td>0</td> <td>8</td> <td>5</td> <td>9</td> <td>7</td> <td>Label</td> </tr> <tr> <td>-</td> <td><i>c</i></td> <td><i>a</i></td> <td><i>b</i></td> <td><i>c</i></td> <td>Pred.</td> </tr> </tbody> </table>	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>	Nodes	*	*	*		*	Compl.	0	8	5	9	7	Label	-	<i>c</i>	<i>a</i>	<i>b</i>	<i>c</i>	Pred.
<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>	Nodes																					
*	*	*		*	Compl.																					
0	8	5	9	7	Label																					
-	<i>c</i>	<i>a</i>	<i>b</i>	<i>c</i>	Pred.																					
<p>The graph is identical to the previous one. Bold arrows indicate relaxation: a to b, a to c, b to d, c to b, c to e, d to c, e to d. Node d is now marked as completed.</p>	<p>Select next closest node to the starting node (node <i>d</i>) and relax all adjacent nodes (bold arrow).</p> <p>If less than the present distance label, update the distance label of the node and the predecessor.</p> <p>Mark node <i>d</i> as completed.</p> <p>All nodes are completed.</p>	<table border="1"> <thead> <tr> <th><i>a</i></th> <th><i>b</i></th> <th><i>c</i></th> <th><i>d</i></th> <th><i>e</i></th> <th>Nodes</th> </tr> </thead> <tbody> <tr> <td>*</td> <td>*</td> <td>*</td> <td>*</td> <td>*</td> <td>Compl.</td> </tr> <tr> <td>0</td> <td>8</td> <td>5</td> <td>9</td> <td>7</td> <td>Label</td> </tr> <tr> <td>-</td> <td><i>c</i></td> <td><i>a</i></td> <td><i>b</i></td> <td><i>c</i></td> <td>Pred.</td> </tr> </tbody> </table>	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>	Nodes	*	*	*	*	*	Compl.	0	8	5	9	7	Label	-	<i>c</i>	<i>a</i>	<i>b</i>	<i>c</i>	Pred.
<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>	Nodes																					
*	*	*	*	*	Compl.																					
0	8	5	9	7	Label																					
-	<i>c</i>	<i>a</i>	<i>b</i>	<i>c</i>	Pred.																					

The distance between the starting node and each of the other nodes (one-to-all) can be read off the final graph or the associated data structure table. The shortest path from each node to the starting node *a* can be retraced by using the information in the last row of the data structure table (named *Pred.*) containing the predecessor to each node. For example, the shortest path from *a* to *d* is 9 units long and is found by retracing starting at the end node: *d* – *b*, *b* – *c*, *c* – *a*. The path is thus *a* – *c* – *b* – *d*. Of course other paths can be found between these two nodes, but all are longer, even if some may have fewer arcs. It must be noted that, as the example is a directed graph (arcs are directional and need not be the same length if bi-directional arcs exist between nodes), the path from *d* to *a* will not necessarily be the inverse of the path from *a* to *d* and is not likely to be the same length.

If the shortest path to a specific node is sought, e.g. from *a* to *e*, it is only necessary to process the algorithm until that node has been marked as completed. The last two steps in the above example would thus not have been required. This can be achieved by terminating the

encompassing *for* loop with a termination condition subject to the required end node having been completed. This can be safely achieved as the algorithm ensures only nodes are marked completed if there is no other uncompleted node that has a shorter distance to the start node.

The solution as implemented for this thesis was a directed graph, which implies that a path is not necessarily the same as its inverse, *i.e.* a path from node  $r$  to node  $s$  is not necessarily the same as a path from node  $s$  to node  $r$ . However, in this application each directional arc has a numerically identical inverse arc and thus the path between start and end-nodes is reversible. This principle of bi-directional arcs between adjacent nodes is shown in Figure 2-7.

The operational scenario that had to be implemented required that additional modifications had to be made to the original Dijkstra's Algorithm. The main departure to the standard implementation was that not only distance (path length) is of importance, but more importantly, that path risk has to be considered. The implication of this is discussed at length in the following paragraph. An additional requirement was that the paths to be found were not necessarily a one-to-one or one-to-all. In real operational scenarios, it is possible for a ship to approach a minefield from any direction and start the penetration anywhere along the leading edge of the minefield. Also, the position of departing from the minefield would, in a fine grid not be restricted to a single node. More likely, is the case where a point of least distance was required within an area. An example of this is where a ship requires passage through a minefield to enter a harbour. Anywhere along the harbour mouth would be a suitable endpoint for the journey. This requirement translates to a many-to-many problem and had to be solved with minimum impact on the solution time of the total problem.

It would have been possible to implement such a solution by sequential application of Dijkstra's Algorithm in a one-to-many version, each time using a different start node as the "one" and a defined subset of the remaining nodes (the end nodes) as the "many". This is a time-consuming method with the solution time being a multiple of the number of start nodes multiplied with the time for a single solution. A more efficient method is the implementation of an algorithm allowing for a many-to-many solution such as the Floyd Shortest Path Algorithm [5] which solves the *all shortest paths* problem.

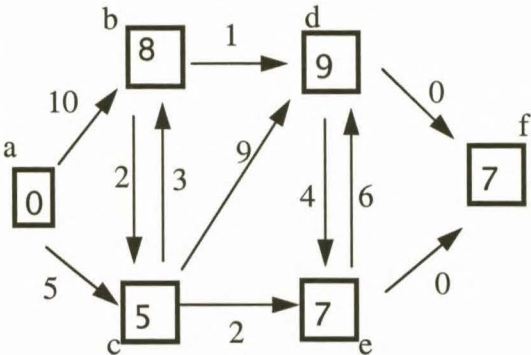


In this analysis, the desired effect was achieved by creating two additional nodes, one for connection to the allowable starting nodes, and the other for connection to the allowed end nodes. Additional uni-directional arcs were created from these two nodes as follows:

- arcs leading from the one new start node to all the allowable start nodes on the grid and
- one arc from each of the allowable end nodes to the second additional node.

The distance value allocated to the new arcs was set as 0. The effect was thus to create a one-to-one Dijkstra application. The principle is shown one-sided in Table 2-2 using the graph and data structure from Table 2-1. The analogy to the described scenario is that the shortest path is sought from node *a* to either node *d* or node *e*. An additional node *f* is created with arcs of length 0 from both node *d* and node *e* directed towards it.

**Table 2-2: Example of modification to Dijkstra's Algorithm to allow multiple end nodes**

Network		Data Structure						
		<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>	<i>f</i>	Nodes
		*	*	*	*	*	*	Compl.
		0	8	5	9	7	7	Label
		-	<i>c</i>	<i>a</i>	<i>b</i>	<i>c</i>	<i>e</i>	Pred.

It is evident that if either *d* or *e* were allowable endpoints, the shortest path would have been from node *a* to node *e*, found by tracing back from node *f* via its predecessor, node *e*. In this solution the problem size is marginally larger than the unaltered graph. At most, two extra nodes are added and the number of arcs that are added is the sum of the allowable start nodes plus the allowable end nodes. This was accepted as efficient enough and no other algorithms were investigated for implementation.

### 2.2.5 *Efficient Frontier Graph*

In the proposed optimisation, it is evident from the problem formulation that the solution is primarily about risk. Additionally it is important that, where risk is considered, very often this has to be done by trading off risk against some other parameter such as cost or its inverse, return.

An area of research parallel to this, where the risk-return relationship is vitally important, is that of financial investments and particularly portfolio selection, where the aim is to obtain the highest return at the lowest risk. The similarity to the path optimisation here is evident. It is likely that the lowest risk is not sought at any cost, but at an acceptable cost. The extreme argument is that all risk could be eliminated by not entering or transiting the minefield at all. This is evidently not acceptable, as the cost of not breaching the minefield is not deemed to be a satisfactory solution to the problem. Therefore the time taken or the distance travelled can be seen as the cost of risk reduction, and the shorter the distance (and hence the less the travel time) the higher the return for an acceptable risk. These arguments have made a formal investigation into the risk-return relationship of different paths an interesting concept.

The principles of the efficient frontier graph were chosen to illustrate these results. It is therefore necessary to give a short background to this area. It has to be noted that the term *graph* here refers to a plot and not to the previously described collection of vertices and edges.

The efficient frontier forms part of the field of *portfolio theory*, also referred to as *modern portfolio theory* and abbreviated MPT. The accepted originator of this is the Nobel laureate Harry Markowitz who described it in a groundbreaking paper in 1952 [14], [15].

The starting point is that a method is sought to express the return or yield of an investment against its risk. Investments are typically based on portfolios which are combinations of different investment vehicles such as equities (shares) and bonds. The *return* is easily measured as the growth of the value of the investment over a period of time. The *risk* is more difficult to express. The chosen measure of *risk* here is the *portfolio volatility* which is measured in the standard deviation of the returns over the same time period that the returns measured. A portfolio which varies widely in value over this period (*e.g.* the weekly share

price when viewed over a year) will have a high standard deviation figure and thus a high associated risk.

If different portfolios are investigated over time, they can be analysed graphically. If the standard deviation of the portfolio is plotted as the  $x$  value and the average return over the same period is plotted on the  $y$  axis, it can be represented as a point on a  $x - y$  plane. If this is done for a number of different portfolios, it can be expected that there will be some variation, and an area on the  $x - y$  plane will be represented by the different portfolios risk-return points. This can be expressed as for the example in Figure 2-5 for portfolios made up of different percentages of three investment vehicles.

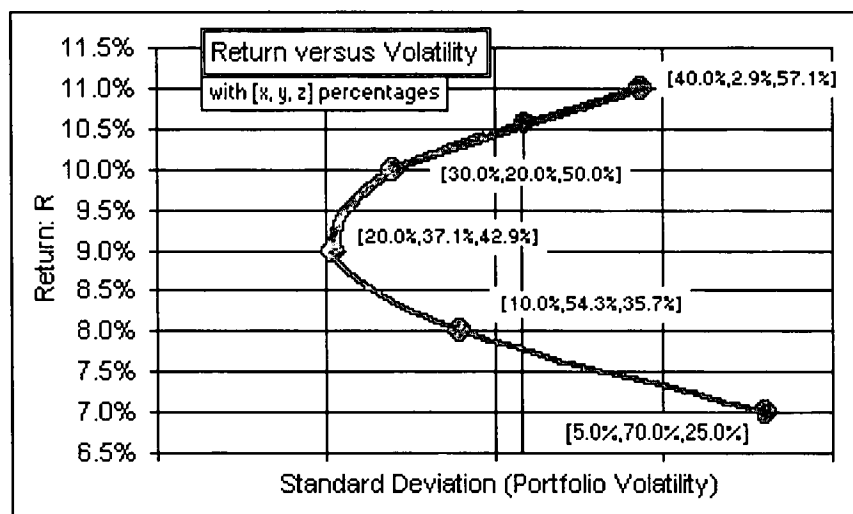


Figure 2-5: The area represented by different portfolios' return *versus* risk for a specified period ([16])

It is immediately clear that the portfolios that are of interest to investors are those with a high return, and particularly those with a combined low risk. A portfolio is termed *efficient* if there is no other portfolio with a better return for a specific risk, or alternatively, no lower risk for a given return. The upper boundary of the portfolio distribution on such a graph forms a frontier, as there can be no portfolio with a higher return for a given risk. In such a case the frontier is termed the *efficient frontier*. This collection of efficient portfolios can be plotted separately and connected by a line, illustrating the best *risk versus return* relationship that can

be expected of a portfolio made up of the represented investment vehicles. Such a graph is shown in Figure 2-6.

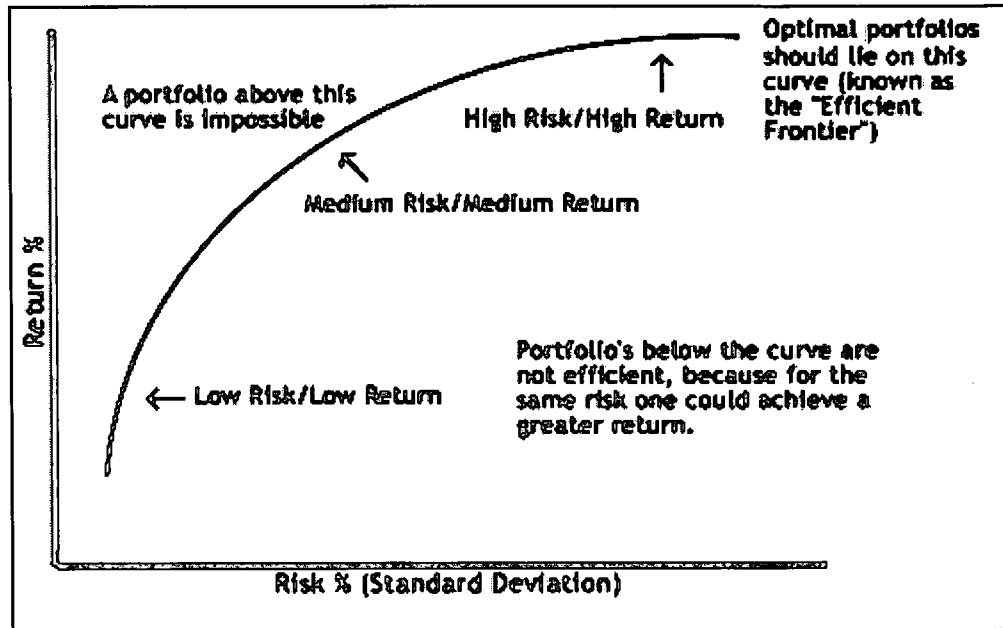
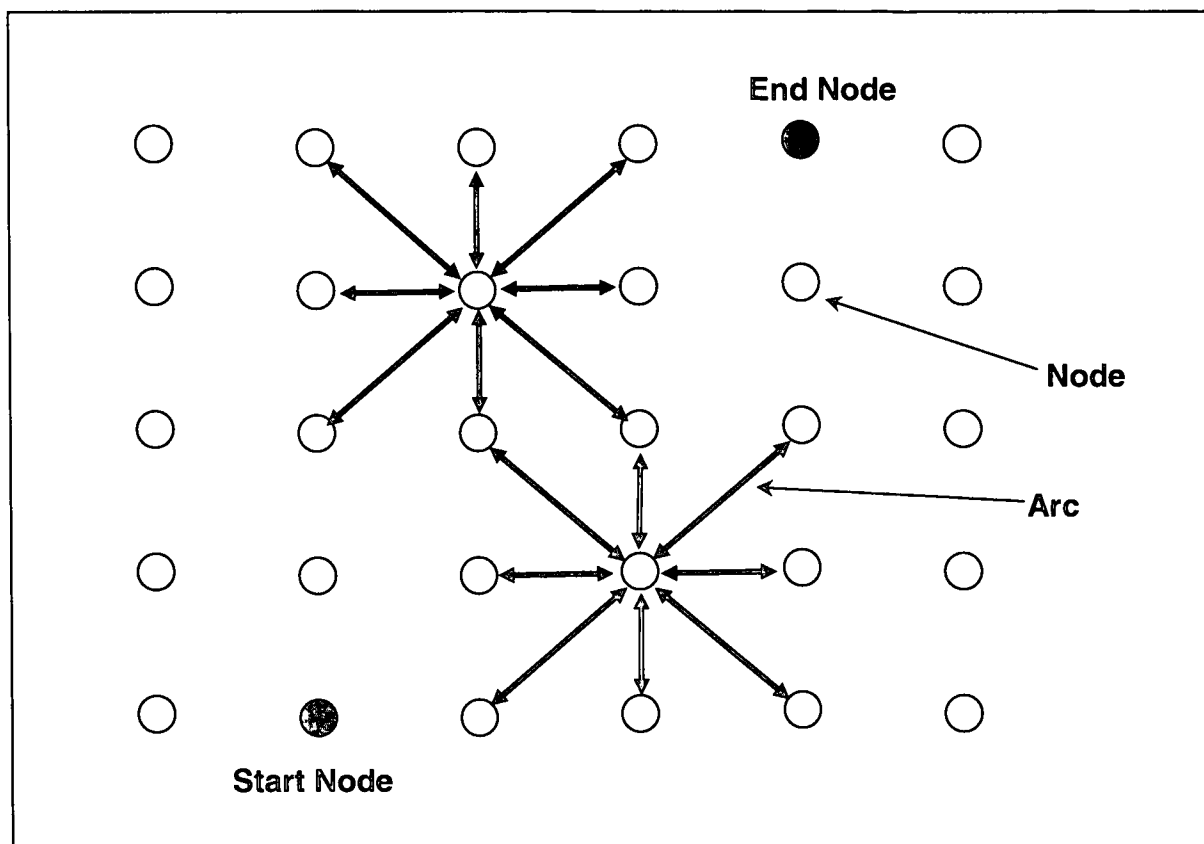


Figure 2-6: Example of an efficient frontier graph ([17])

The topics *Graph Theory* and *Dijkstra's Algorithm* which have been dealt with in theory here, are shown in their implementation in the following paragraphs. The *Efficient Frontier Graph* implementation is addressed in the paragraph detailing the implementation results.

### 2.2.6 General description of the solution implementation

The area containing the mines is transformed into a network by dividing the area of interest into a finite number of geometrically spaced nodes. These nodes are interconnected by arcs along which travel is possible. Only adjacent nodes are connected to each other in order to keep the total number of arcs sufficiently restricted for a manageable problem size. Travel is possible along the arcs in either direction, hence arcs are bi-directional. This is shown graphically in Figure 2-7 for two selected nodes.



**Figure 2-7: The routing structure and some nomenclature**

Next, the effect of the mines is included. This is achieved by adding their position and, if required by the user, that of their chosen safety radius. In such a case, all nodes and connected arcs within the resulting safety circle are removed as they represent travel through areas of unacceptable risk. This is implemented by erasing the nodes that fall within the safety circle and because of that, their associated arcs. This is shown graphically in Figure 2-8.

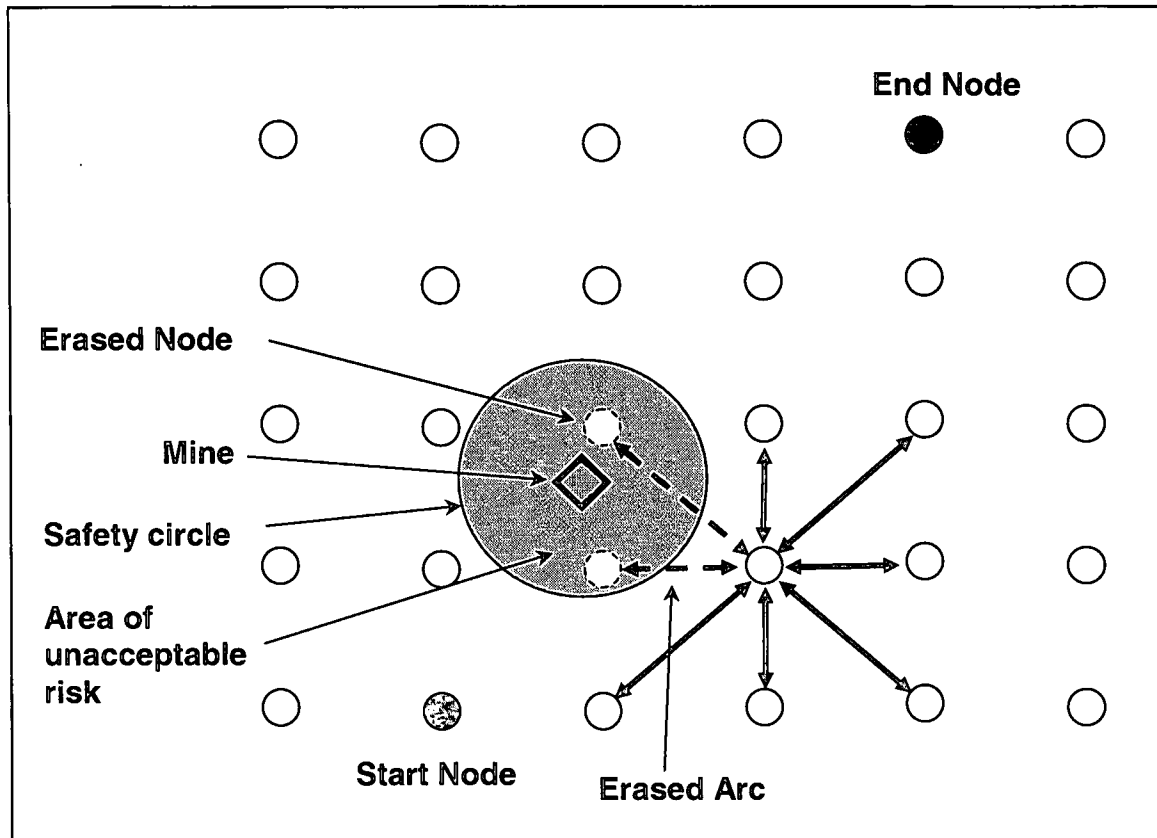
A risk figure is allocated to each remaining arc as a function of the arc's proximity to the mines. This risk figure can be determined as preferred by the person doing the analysis. Options are:

- a function of the closest point of approach to the *closest* mine, or
- a function of the closest point of approach to *all* mines.

Both are described in paragraph 2.2.7 (p. 23).

Finally, a method has to be used with which the best combination of arcs is selected so that the path with the smallest total risk is chosen. The problem to be solved is that of a *single*

*source shortest path*. This is achieved with the previously described shortest path algorithm, Dijkstra's Algorithm, where the described arc cost function becomes a function of mine proximity and/or the arc's length. The arc cost function is central to Dijkstra's least cost optimisation process and is described in the following paragraph.



**Figure 2-8: The concept of mines with safety radius**

### 2.2.7 Quantification of arc cost function in terms of risk

Since the path optimisation is required to be based on risk, it is a requirement that the arcs that have to be selected, have a quantified risk figure in lieu of the length that is associated with them for conventional Dijkstra's algorithm optimisation. The principle of arc weighting based on parameters other than length is generally accepted and referenced, for example in [18] where vehicle routing in a road network is done with factors such as sector length, gradient and allowable travelling speed. Furthermore, the concept of weighted regions is introduced in [8] for the more directly related case of terrain navigation where Dijkstra's or A\* algorithms are used for generating minimum cost paths. Risks are associated with different regions based on their features and the risk "cost" is minimised to find the best path.



For the situation where a ship has to pass a mine, it intuitively seems correct to couple risk to the distance between the two, expressed as:

$$\text{Risk} = f(R) \quad (1)$$

where  $R$  is the distance between ship and mine.

It seems obvious that the greater the distance between the two, the lower the risk ought to be. For the purpose of this thesis it is necessary to quantify the risk of the total path so that the path with the smallest accumulated risk is selected. The starting point to achieve this is by applying risk values to all arcs so that the combination of arcs that give the lowest total risk can be found with Dijkstra's Algorithm. Boerman [9] presents two different criteria to establish the relationship between distance and risk which are summarised here.

The first premise is that risk is coupled to the probability of a ship detonating a mine (abbreviated as  $PD$  for *Probability of Detonation*). This is important where it is required that the ship does not detonate a mine even at a long distance, so that the exploding mine does not notify the enemy. This is especially relevant to a submarine transiting an area, as stealth is its main feature. The relationships that govern if a mine detects a ship are based on the strength of the ship's signatures received at the mine. These signatures are primarily its acoustic and magnetic characteristics, but pressure may also play a part. It is argued that these signatures are related to the inverse of the distance squared. With travel along a unit distance, the relationship can be approximated by  $a/R_m^2(\delta)$  with  $a$  a constant and  $R_m(\delta)$  the distance between the mine  $m$  and a unit of arc along which the travel takes place. When starting at 0 and travelling along an arc  $\delta'$  long, the probability of detonating mine  $m$  can be expressed as:

$$PD_m = \int_0^{\delta'} \frac{a}{R_m^2(\delta)} d\delta \quad (2)$$

If this were implemented for a discrete case where the arc is divided into  $z$  sections, each with a length of  $l_z$  and  $R_{mz}$  is the minimum distance between the mine  $m$  and arc segment  $z$ , the above equation can be approximated as:

$$PD_m = \sum_z \frac{al_z}{R_{mz}^2} \quad (3)$$

When applied to an arc  $(i,j)$ , a worst case is that:

$$PD_{mij} = \frac{a D_{ij}}{R_{m \min}^2} \quad (4)$$

$PD_{mij}$  is the probability that a ship traversing arc  $(i,j)$  will detonate mine  $m$ ,  $D_{ij}$  is the arc length and  $R_{m \min}$  is the nearest point on the arc  $(i,j)$  to mine  $m$ .

The second premise is that risk depends on the probability that a detonated mine will damage the ship. The likelihood of damage to a ship depends on the amount of energy transferred from the mine detonation to the ship and is expressed in terms of a shock factor:

$$SF = \frac{\sqrt{W}}{R} \left( \frac{1 + \sin \phi}{2} \right) \quad (5)$$

where  $W$  is the mass of the explosive (in equivalent of lbs TNT),  $R$  the slant range between the explosion and ship (in imperial feet) and  $\phi$  the angle between the line from explosion to the ship and the tangent to the hull at the point nearest to the explosion [19]. In reference [9]  $\phi$  is described as the angle between the ocean surface and the line from the explosion to the nearest part of the hull. The difference in definition is immaterial here, as in both the relationships the shock factor SF (and thus the magnitude of damage) is a function of  $1/R$ .

Equation (4) implies that two arcs with lengths  $D_{ij}$  and  $2 D_{ij}$ , respectively and for which  $R_{m \min}$  is equal, will have a  $PD$  differing with a factor of two. This reasoning is not agreed with here because it is not believed that the probability of detection is influenced to such a magnitude by the arc length  $D_{ij}$ . There is merit in the argument that for two arcs of differing length, a higher risk should be allocated to the longer arc, as the ship spends more time in the vicinity of the mine than in the case of a shorter arc. Whether this risk is proportional to the arc length is not known, but for the purpose of this thesis it is accepted as such. The argument here will be put that risk is a function of the distance between the ship and the mine, but depending on the preference of the decision maker, can be selected as proportional to  $1/R$  or  $1/R^2$  or indeed any other power of  $R$ . For that reason the risk factor has been made user



selectable in the computer application and the effect of this on path selection and path risk was evaluated in paragraph 2.5.5 (p. 49). Thus it is stated that for an arc  $(i,j)$ :

$$RiskCost_{ij} = \frac{ArcLength_{ij}}{R_{min}^k} \quad (6)$$

with  $ArcLength_{ij}$  the length of the arc  $(i,j)$ ,  $R_{min}$  the distance of closest approach to the *closest* mine and  $k$  a user selectable preference, but likely to be either 1 or 2. This causes the unit associated with risk to vary, as arc length is in a unit of length, and depending on the value of the exponent  $k$ , its unit will be  $LengthUnit^{1-k}$ . It was considered to introduce a factor into the equation to obtain a dimensionless value for arc risk cost, but as this was deemed unimportant to the principles under examination in the thesis, it was discarded. Thus in all the graphs and results of path risk, the unit shown with the risk values gives an indication of the  $k$  value of the associated arc risk cost function.

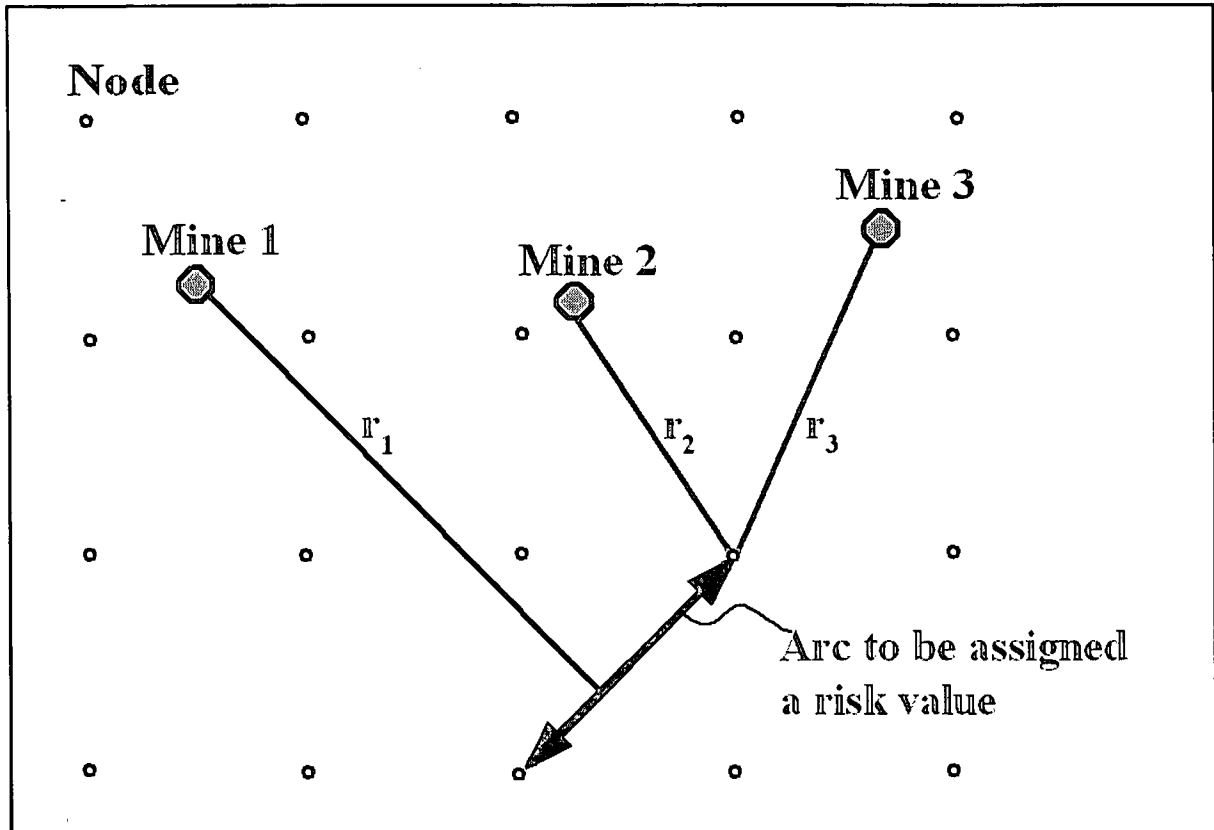
There is a further argument that the risk of a particular arc may not depend only on the closest mine to it, but in fact on the effect of all the mines. Naturally, the closest mines will still have the greatest effect on the risk value, but for the example of the hypothetical case where there are two mines with equal closest point of approach to the arc. The risk associated with such an arc will be twice as high as for a single closest mine. For that purpose, the implementation has been made user selectable, determined by whether the decision maker wants to evaluate the risk of all mines or the risk of the closest mine to the arc only. The effect of this has been reported in paragraph 2.5.4 (p. 48). For the case when all mines are being considered for risk evaluation, the risk of an arc  $(i,j)$  is:

$$RiskCost_{ij} = ArcLength_{ij} \times \sum_{i=1}^n \frac{1}{R_{i\ min}^k} \quad (7)$$

The factors are similar to those in equation ( 6 ) but with  $R_{i\ min}$  the distance of closest approach to *mine i* and  $n$  the number of mines in the minefield.

In Figure 2-9 the principle of an arc being assigned a risk value as a function of three mines is graphically shown. In the case of *Mine 1* in that figure, the shortest distance is perpendicular to the arc. For the other two mines, it is the distance between the node forming the end of the arc and the mine position that gives the shortest distance. Thus, for the case when only the closest mine is considered, the assigned risk will be a function of  $r_2$ . If the risk is chosen to be

determined by all mines, it will be a function of  $r_1$ ,  $r_2$  and  $r_3$ . It must be noted that the scale in the figure is not representative of an operational implementation, as the spacing between nodes needs to be substantially smaller to achieve an acceptable solution. The issue of grid spacing is analysed at length in paragraph 2.5.1 (p. 35).



**Figure 2-9: Assigning a risk to an arc as a function of the distance to the closest mine**

As will be seen later in the results section, there are instances where the paths are required to be optimised with regard to path length. For this purpose, and that of evaluating the path lengths, arcs need a second cost function which is based on their length. This second attribute of an arc is taken to be its length only. In such cases arc cost function is equated to arc length. Thus for an arc  $(i,j)$ :

$$\text{LengthCost}_{ij} = \text{ArcLength}_{ij} \quad (8)$$

It will furthermore be shown that there is also a requirement to adjust the optimisation to ratios between the two extremes of shortest path and safest path. This necessitated a cost

function that takes into consideration both arc risk and arc length. If this is done, a risk weighting can be applied to determine paths with different ratios of optimisation for risk and length. If maximum risk is acceptable, the *shortest path* will result, if minimum risk is required, the *safest path* results, taking only arc risk into consideration. As the two arc attributes do not have the same unit, simple addition of the two attributes to determine a single cost attribute for an arc is not possible.

Combining the two dissimilar attributes into one value could be best achieved by linking them to a primary operational objective so that there is a direct relationship between path risk and that objective as well as path length and the objective, so that:

$$\text{OpsObjective} = f(\text{PathRisk}, \text{PathLength}) \quad (9)$$

As the development of such a relationship is complex and dependent on the operational scenario, it was deemed to fall outside the scope of this thesis. Instead, an approach was followed where the risk and length attribute of an arc would be expressed in a dimensionless way by transforming it into a ratio relative to the average of all arcs in the grid. This is easily implemented because as is evident from the simplified implementation flow diagram in **Figure 2-10** (p. 31), all arc lengths and risks are determined before the optimisation with Dijkstra's Algorithm is implemented. These ratios could now be used in the final cost function of any arc, so that the optimisation function which was minimised in Dijkstra's Algorithm is the function of the two attributes of arc  $(i,j)$ :

$$\text{ArcCost}_{ij} = (1 - \text{RiskWeighting}) \times \text{RiskRatio}_{ij} + \text{RiskWeighting} \times \text{LengthRatio}_{ij} \quad (10)$$

with,

$$\text{RiskRatio}_{ij} = \text{RiskCost}_{ij} / \text{MeanRiskCost} \quad (11)$$

where 
$$\text{MeanRiskCost} = \frac{1}{n} \sum_{i=1}^n \text{RiskCost}_i$$

and

$$\text{LengthRatio}_{ij} = \text{LengthCost}_{ij} / \text{MeanLengthCost} \quad (12)$$

where 
$$\text{MeanLengthCost} = \frac{1}{n} \sum_{i=1}^n \text{LengthCost}_i$$

for  $n$  the number of arcs in the entire grid.

*RiskWeighting* (in equation (10)) is a user selectable value between 0 and 1. It is evident that selecting 0 as *RiskWeighting* will result in the *safest path* by optimising for path risk only, and selecting 1 will result in the *shortest path* by optimising for arc length only.

It must be noted that while the optimisation is done with a cost function that uses modified values of the *RiskCost* and *LengthCost*, by multiplying them with a constant of their mean and risk weighting, the original values are retained as parameters that characterise the arc. These original values are used in determining the total path risk and path length made up of a path's arcs as will be discussed in § 2.2.8 and § 2.2.9 respectively.

Mathematically the principle of multiplying an arc cost attribute with a constant (here the constant is the inverse of the mean *RiskCost* or *LengthCost* and the risk weighting) does not influence the path optimisation algorithm as the constant is equally applied to all arcs. The simplified equivalent could be shown in the example used to demonstrate Dijkstra's Algorithm and which is shown in Table 2-1 (p. 15). There, if all the arc lengths were multiplied with a factor of 10, the same shortest path would result in terms of the sequence of nodes. The total path length would naturally increase with a factor of 10.

#### 2.2.8 Quantification of path risk

Any path through the minefield consists of a combination of connected arcs. The risk value of that path is a function of the risks of the arcs that make up the path. Thus, for a path of  $n$  arcs between two nodes  $r$  and  $s$ , the associated risk is

$$PathRisk_{rs} = \sum_{i=1}^n RiskCost_i \quad (13)$$

where  $RiskCost_i$  denotes the risk associated with the  $i$ -th arc in the path from node  $r$  to node  $s$ .

#### 2.2.9 Quantification of path length

Similarly, the length of that path is a function of the length of the arcs that make up the path. Thus, for a path of  $n$  arcs between two nodes  $r$  and  $s$ , the associated length is

$$PathLength_{rs} = \sum_{i=1}^n LengthCost_i \quad (14)$$

where  $LengthCost_i$  denotes the length of the  $i$ -th arc in the path from node  $r$  to node  $s$ .

### 2.2.10 Introduction of safety radius

Here the principle of a safety radius, briefly expounded in § 2.2.6 is described. During the initial implementation of the principles described above, safe paths resulted that were obviously optimised for total path risk, but there remained some situations where it was evident that they passed unacceptably close to closely spaced mines. This necessitated the introduction of the principle of a safety radius which is graphically shown in Figure 2-8. This is a circle around each mine into which the ship was not allowed, either because unacceptable damage would result to the ship, or the probability of detonating a mine would be too high. Practically, this was introduced by investigating which nodes fell into the respective safety circles and deleting them. This in turn resulted in the removal of all arcs leading to or from the deleted nodes, reducing the total number of arcs and forcing paths away from such areas. If these circles of adjacent mines intersected, passages between mines that were previously possible were effectively closed off. The net expected result would thus be paths with a higher total risk, but a lower maximum local risk. An improvement by optimising for this *bottleneck problem* is proposed in 4.3.1 (p. 80).

## 2.3 Computer implementation

The above principles were combined in a computer-based application with a simple user interface. The basis used for this was the library-based modelling and simulation environment *Extend® V6* produced by the American company ImagineThat Inc. It is a tool that falls in the same broad category as *Simul8* and *Arena*. It is capable of discrete-event and continuous modelling. A feature is that the blocks in its libraries can be modified by the user or additional blocks can be created by Extend's® own programming language structured similarly to C and named ModL.

The structure of the code generated for the optimisation is described in a simplified flow-diagram in Figure 2-10. The block containing the optimisation implementation is at the centre of the part named "Model" in the diagram of the complete Extend® implementation shown in

Figure 2-11 and in detail with its user interface in Figure 2-12. It consists entirely of code generated for that purpose by the student (APPENDIX A). The other parts of the model are mostly components from the Extend<sup>®</sup> standard libraries consisting of the plotter, file input and output for running batches where changing inputs are required and the results need to be saved. The additional two blocks are custom blocks to generate minefields (random, mine-lines or custom positions) and to provide a starting trigger for the optimisation execution. They are not further described as they are not central to the optimisation described in the thesis.

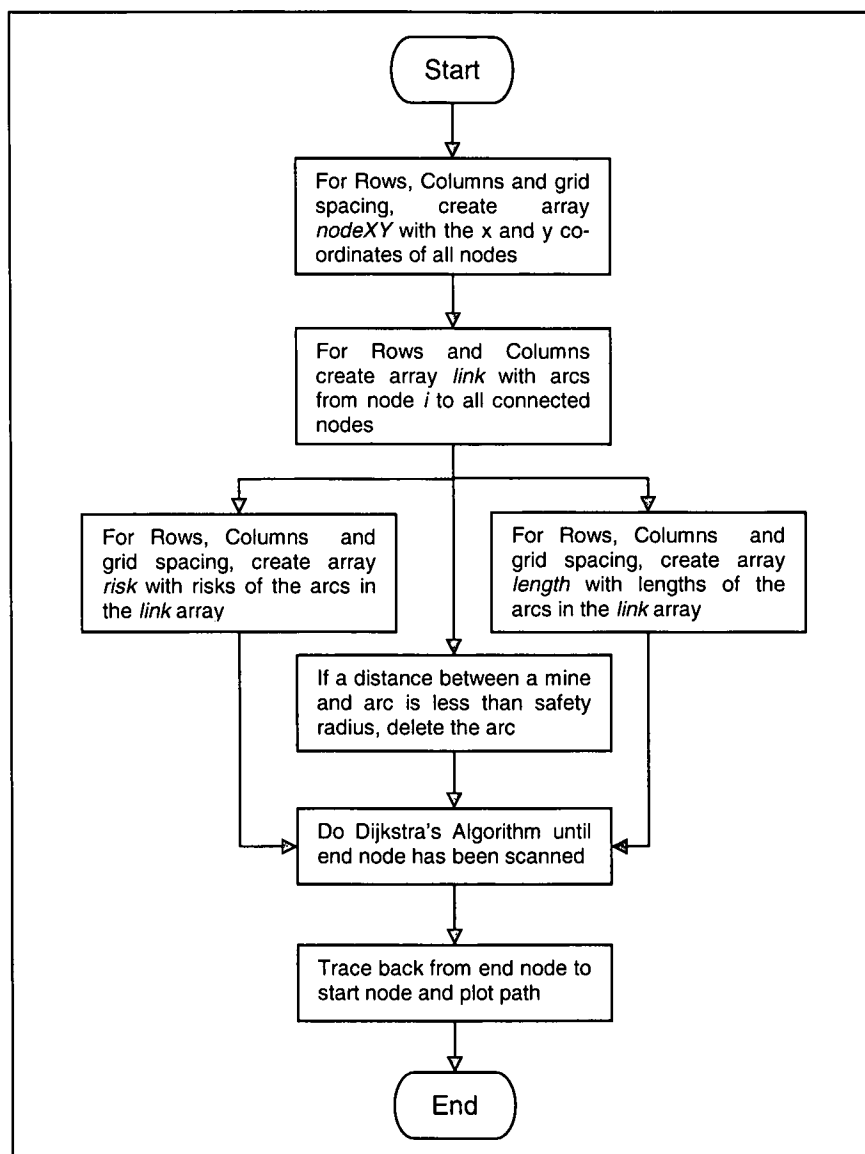


Figure 2-10: Simplified flow diagram for Route Optimiser block



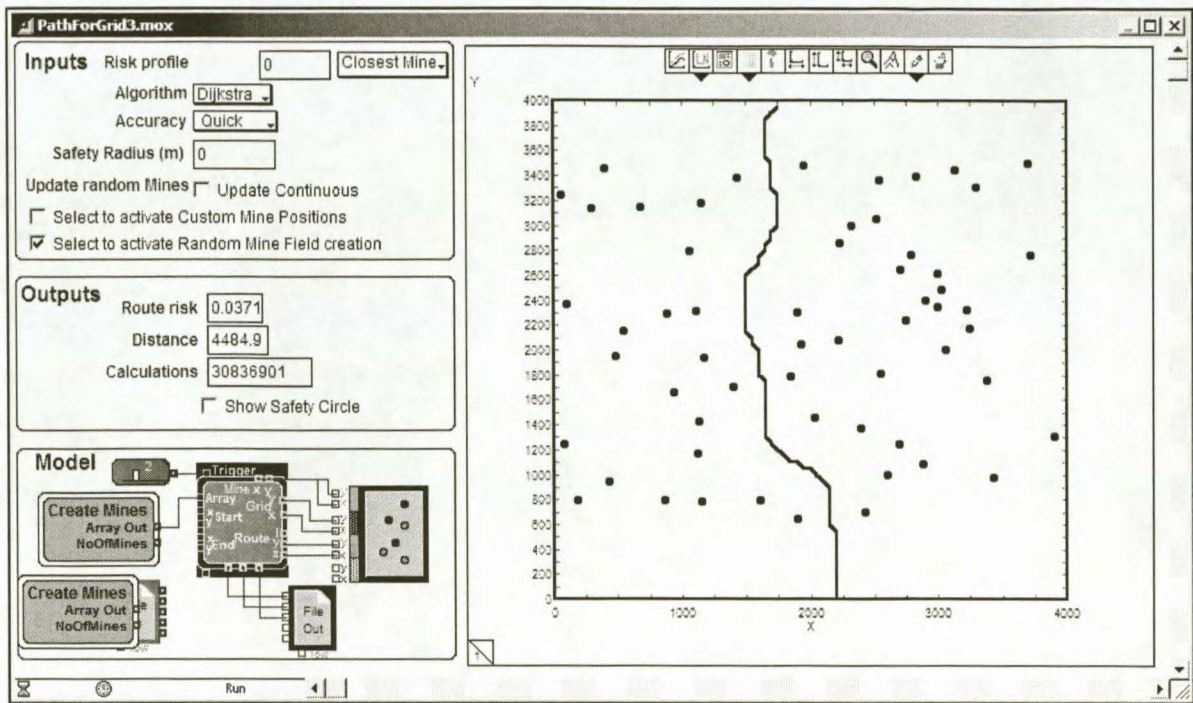


Figure 2-11: Extend® model elements and reduced user interface for path optimisation

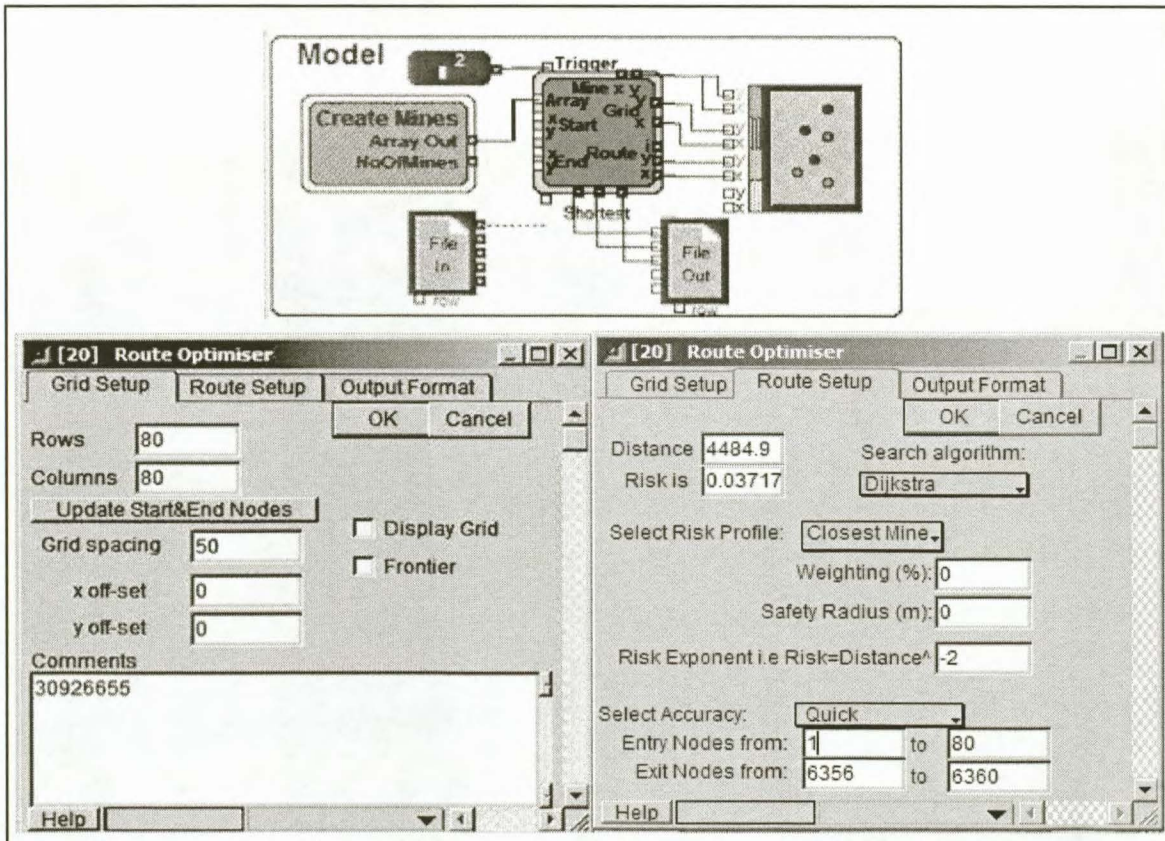


Figure 2-12: Route Optimiser block detail and important user interface

## 2.4 Parameters of importance to the implementation

In this section the parameters are described that influence the methodology implementation. They are listed in the same order as in Section 2.5 (p. 34) which presents the results and discussion of the parameters that were investigated. The issues of the minefield pattern (random versus structured) and number of mines in the minefield (density) are viewed as input parameters to the problem formulation. As they do not have an effect on the solution implementation (although they influence individual results), they are not included in the discussion.

### 2.4.1 *Grid spacing*

Grid spacing is central to the discretised solution as described in this thesis, as it determines the accuracy of the path that is generated by the simulation. A finer resolution will generally provide improved accuracy, but at the cost of solution time. As the solution time depends on the number of arcs (approximated as 8 additional arcs for each additional node) and the node number increases as the inverse square of the grid spacing *i.e.* if grid spacing halves, the nodes increase approximately fourfold (three extra nodes) and consequently the arcs increase thirty two-fold. The actual figure is slightly smaller as the edge nodes are only connected by 5 arcs and the corner nodes connected by 3 arcs to adjacent nodes and the number of rows and columns does not exactly double with halving the grid spacing as the outer boundaries (which form the outer rows and columns) stay fixed. If there are  $n$  rows and  $n$  columns in the grid, there will be  $2n-1$  of each after halving the grid spacing.

### 2.4.2 *Arc risk cost function*

The equations ( 6 ) and ( 7 ) (p. 26) determine the risk cost function for arcs. The first parameter to be considered is the risk exponent  $k$  which has to be selected depending on the physical relationship between mine proximity and its resulting risk. In addition, the choice whether to make arc risk a function of the closest mine only, or as a function of all the mines in the minefield must be made.

### 2.4.3 Safety radius

A safety radius can be introduced to ensure no path passes more closely than at a predetermined distance to a mine. This is especially relevant in the case where paths are optimised for distance.

### 2.4.4 Risk weighting

In the event that paths are required which are not optimised for the two extremes of *smallest path risk* or *shortest path length*, a risk weighting has to be introduced which combines elements of the two attributes to provide an objective function manipulated by a risk weighting. As used here, it is used to generate paths with varying combination of risk and length. These paths can be evaluated to choose a suitable balance between the two attributes.

## 2.5 Sample results for path optimisation

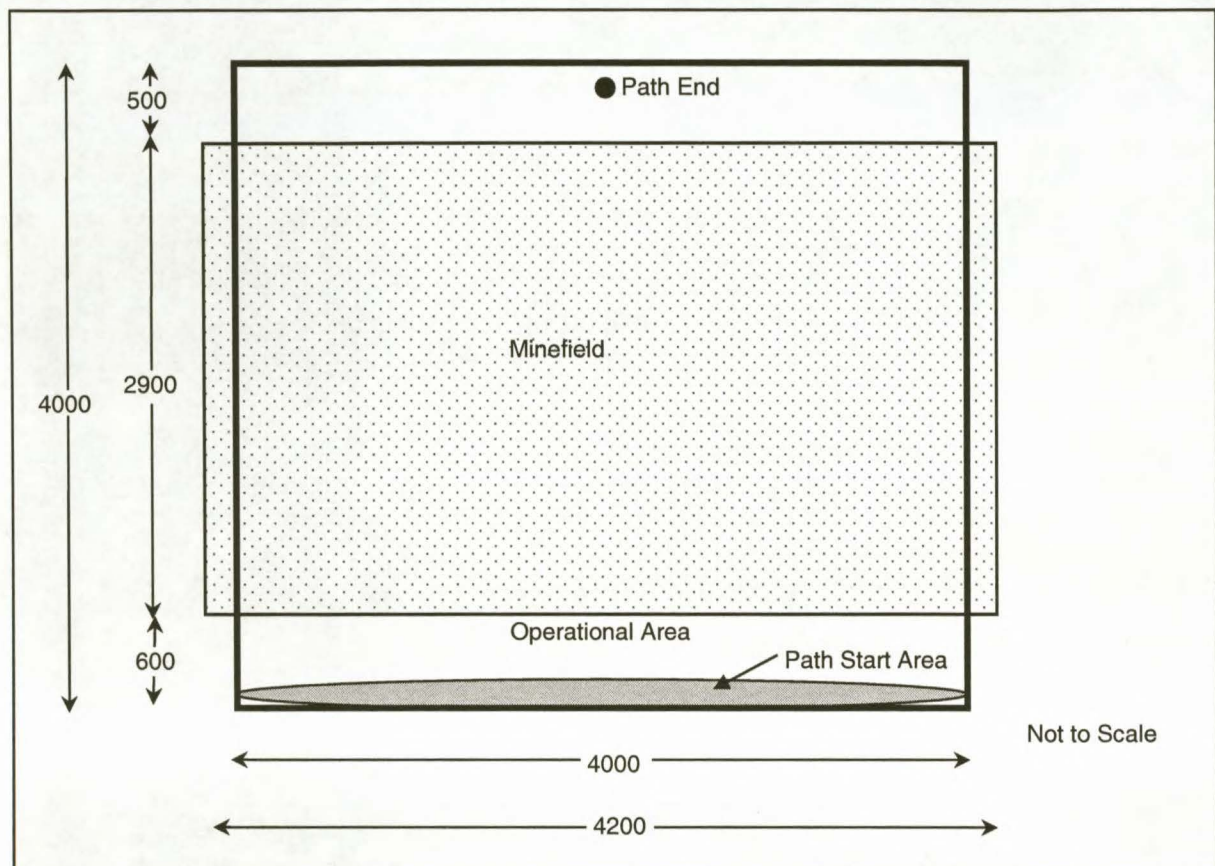
Below are a range of results for some minefields. The parameters used are the following, unless specifically stated otherwise:

- **Number of mines in the minefield:** 60
- **Minefield pattern:** randomly generated
- **Grid spacing:** 50 meters
- **Risk exponent  $k$**  (refer to equations ( 6 ) and ( 7 ) p. 26): 2
- **Risk-profile:** Arc risk is a function of the closest mine only.
- **Safety radius:** 0 meters
- **Path start and end:** Paths in each case commence along the bottom row of the operational area and terminate in the centre of the top row of nodes. This represents a ship arriving from the open sea (the bottom row of nodes) and passing through a minefield to enter a restricted area such as a harbour or channel (the centre of the top row of nodes). The path for a ship leaving a harbour and travelling to the open sea is given by the same path.

The operational area was chosen to be 4000m x 4000m. The area where the mines are deployed (the minefield) was arranged in a way to allow a slight overlap of the left and right edges of the operational area to prevent skirting the minefield by travelling around it. To ensure that mines are not too close to the allowed start and end nodes, bands below and above



the minefield were kept free of mines. The described layout is graphically represented in Figure 2-13.



**Figure 2-13: Layout of operational area and minefield for the displayed results**

The appropriate sub-paragraphs below (2.5.1 - 2.5.10) address the important parameters to be selected when applying the suggested implementation in the way described. It is accepted that the exact representation of the minefield will influence the detail of the results, but the output here will clearly demonstrate the trends.

### *2.5.1 Grid spacing for a specific minefield*

For an area of fixed size, the grid resolution determines the number of nodes and from that the number of arcs. Thus with increasing grid resolution, it is expected that the quality of results will improve, but at the cost of increasing time to reach the solution. It is expected that there is some optimum point where increasing grid resolution will not bring justifiable improvement in the quality of the solution.

To investigate the behaviour of the optimisation with changing grid spacing, the following approach was taken:

- The coarsest feasible grid spacing was selected as 400m spacing, giving for the 4000m x 4000m operational area,  $4000/400+1=11$  rows and 11 columns of nodes for a total of 121 nodes (see grid in Figure 2-14 (a)).
- Next, additional rows and columns were inserted between the existing rows and columns, making it a total of 21 rows and 21 columns (this equals a total of 441 nodes). This results in a grid spacing of 200m and is shown in Figure 2-14 (b).
- This is continued, each time inserting rows and columns by halving the spacing between the previous set of rows and columns to create 41, 81 and 161 rows and columns respectively for a grid spacing of 100, 50 and 25m.

The above approach was chosen to ensure that a simulation of the next finer resolution has the previous nodes available, as well as an additional node between it and its adjacent node. This is shown graphically in Figure 2-14. The effect of *grid spacing* on *path risk* and *path length* was subsequently investigated. The outputs in Figure 2-14 and the accompanying Table 2-3 give an indication of grid resolution *versus* some of the results which may be considered by the decision maker using this tool. The result for the finest grid resolution (25m spacing) is not shown graphically, as the plotted nodes obscure output detail and the path does not change noticeably. The scenario described here is for a ship seeking a path through the previously defined minefield. The fine dots in Figure 2-14 indicate the position of the nodes, while the heavy dots indicate the position of the mines. The user input for all plots is set up for the ship to enter at any point along the lower boundary and leave at the centre node of the top row.

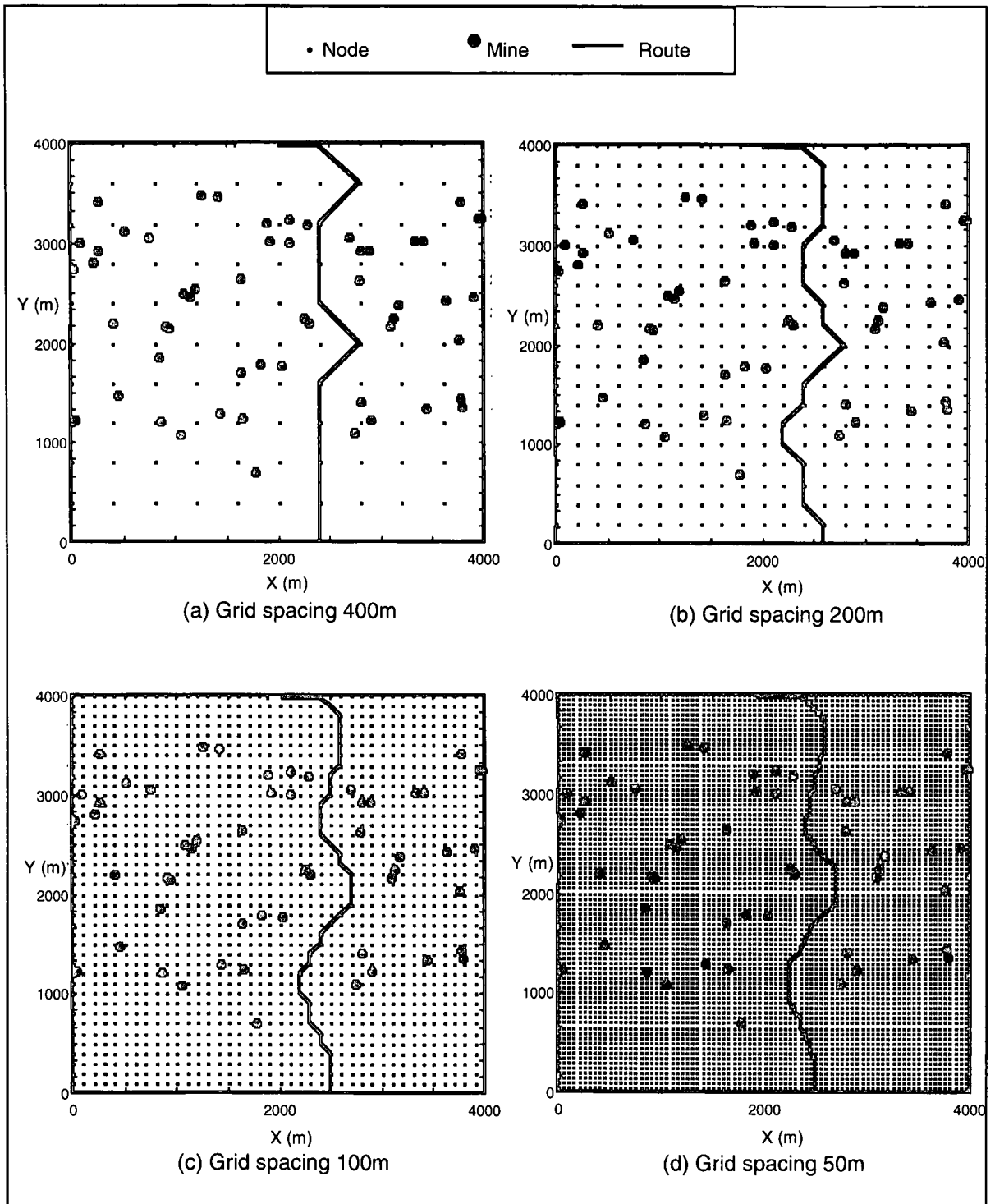


Figure 2-14: Paths generated using different grid spacing (detail in Table 2-3)

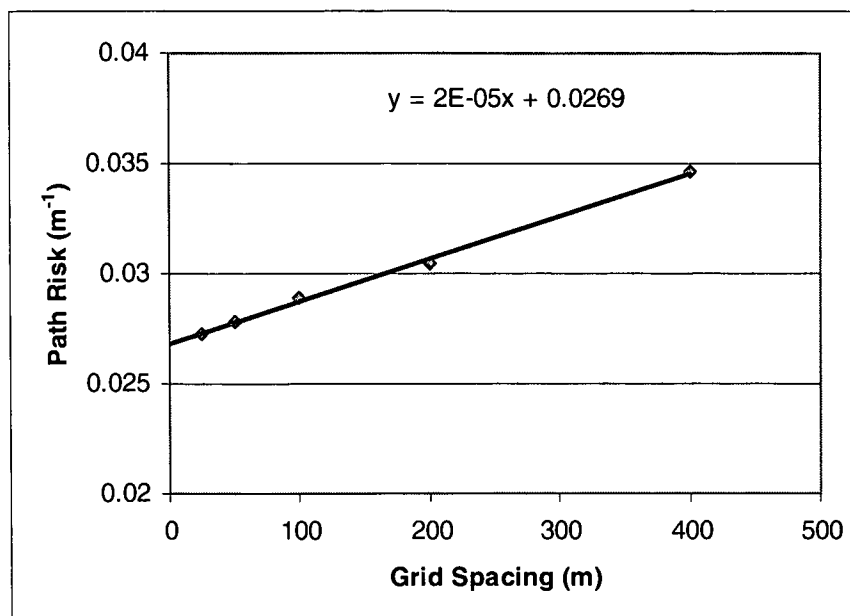


**Table 2-3: Summary of effect of grid spacing on optimisation output**

Figure	Grid Spacing (m)	Path Risk ( $m^{-1}$ )	Path Length (m)	Loop Iterations	Solution Time (seconds)
Figure 2-14 (a)	400	0.0347	5463	10287	0.3
Figure 2-14 (b)	200	0.0305	5346	128359	0.9
Figure 2-14 (c)	100	0.0289	5121	1812123	3.6
Figure 2-14 (d)	50	0.0278	5001	27247503	21.8
Not shown	25	0.0272	5001	430497123	229.0

As the ultimate output and result of the routing solution is an optimised path, the purpose of the discussion in this paragraph is to investigate the effect of grid spacing on model performance and to determine how close to optimal the resulting solution is. The most important aspect of this performance is the accuracy of the result. This accuracy cannot be determined precisely, as the true result is not analytically computable and is therefore not available here. The model gives an approximated result due to the inaccuracies arising from the discretisation (*i.e.* paths are made up of discrete arcs between adjacent nodes only). The coarser the grid, the worse the result is expected to be. The closest possible answer to the optimalcorrect one, would be obtained with a grid spacing of was zero, *i.e.* an infinitely fine grid. Although this is not possible to obtain, and even approaching it would take extreme solution time, a value can be obtained from extrapolating the graph in Figure 2-15 to obtain an approximated actual risk. It is used here to get a performance reference value to determine the model accuracy. By doing regression analysis of the data points in the graph, the value is estimated as  $0.0269m^{-1}$ . Accuracy of the other grid spacing results is calculated from this value as a percentage deviation from  $0.0269m^{-1}$  according to:

$$Accuracy(\%) = 100 * \left( 1 - \left( \frac{PathRisk - 0.0269m^{-1}}{0.0269m^{-1}} \right) \right) \quad (15)$$

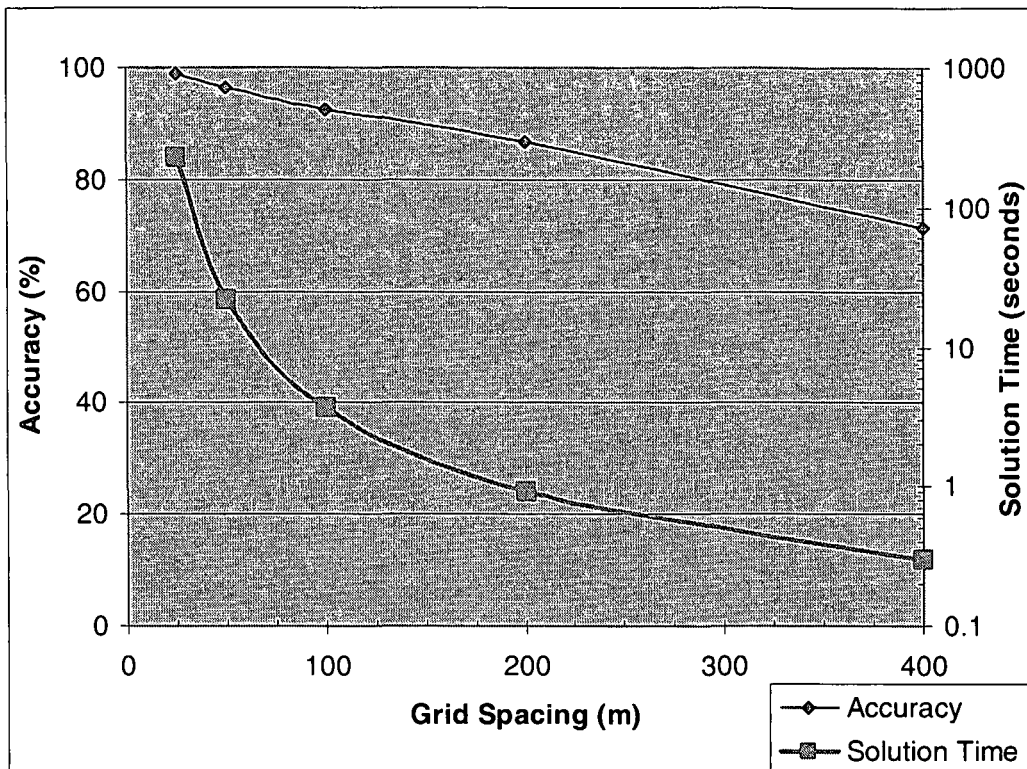


**Figure 2-15: Path risk *versus* grid spacing**

The results of the accuracy calculated for each of the investigated instances is displayed in Table 2-4. This accuracy was then plotted as a function of grid spacing and is displayed in Figure 2-16. The solution time has also been shown, plotted on a logarithmic scale.

**Table 2-4: Result accuracy using  $0.0269\text{m}^{-1}$  as path risk reference**

Grid Spacing (m)	Path Risk (m <sup>-1</sup> )	Accuracy (%)	Error (%)
400	0.0347	71.2	28.8
200	0.0305	86.7	12.3
100	0.0289	92.4	7.6
50	0.0278	96.5	3.5
25	0.0272	98.8	1.2



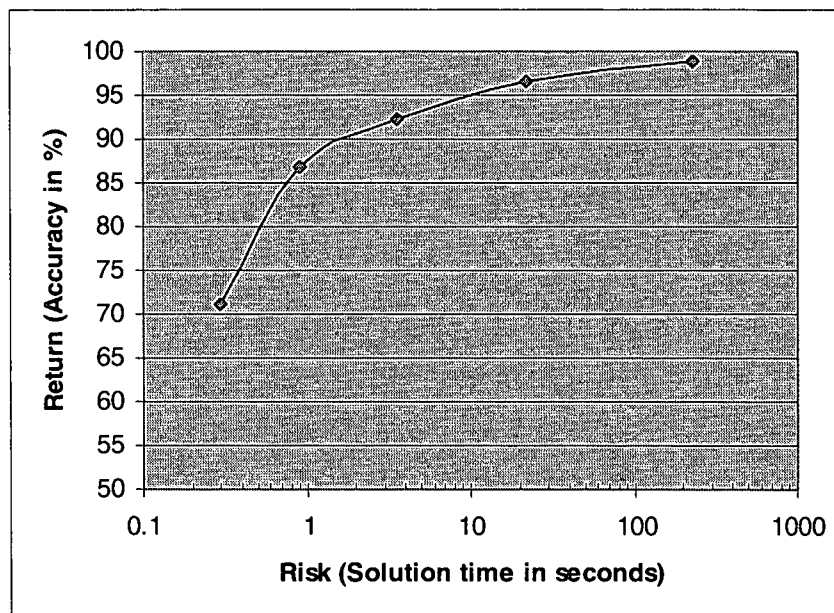
**Figure 2-16: The effect of grid spacing on output accuracy and solution time**

It can be seen that an incorrect path may be generated if the grid is not fine enough (Figure 2-14 (a)). With finer grid spacing the path becomes more accurate (Figure 2-16) because of the increasing number of arcs available. It can be seen that at some stage the increase in accuracy, while still improving with finer grid spacing, comes at the expense of a large increase in computing effort and resulting solution time. Next a method is presented with which a decision can be made at which stage the time increase is excessive with regard to the additional accuracy benefit.

### 2.5.2 *Selection of a suitable grid spacing*

Figure 2-16 clearly shows the relationship between *result accuracy* and *solution time*, but both are plotted against grid spacing. It would be desirable to know what their relationship to each other is, and that requires a dedicated graph so that conclusions can be drawn. While attempting this, it was decided to present it in the form of an Efficient Frontier Graph which is a valuable aid in expressing the relationship between risk and return, especially with regard to investment portfolios. Here the principle is used to guide the user in selecting a suitable grid spacing to obtain the desired accuracy. The main objective is that a correct path is generated

and this has to be ensured by choosing a sufficiently fine resolution. If solution time is not an issue, the finest possible resolution can be chosen. However, time to find a solution is normally not unrestricted and this is thus taken as the “risk”. It can be seen from Figure 2-16, where the secondary y axis is plotted logarithmically, that the solution time increases in excess of exponentially. Hence it is plotted logarithmically in Figure 2-17, but this time it is regarded as the “risk” and represented on the x axis. The “return” is the resulting accuracy and is shown in Figure 2-17 on the y axis. The decision maker can use this figure to determine a suitable trade-off between “risk” and “return” by weighing up the solution time against the resulting accuracy. Once the desired accuracy is selected for an acceptable solution time, it can be determined from Figure 2-16 what the required grid spacing is.

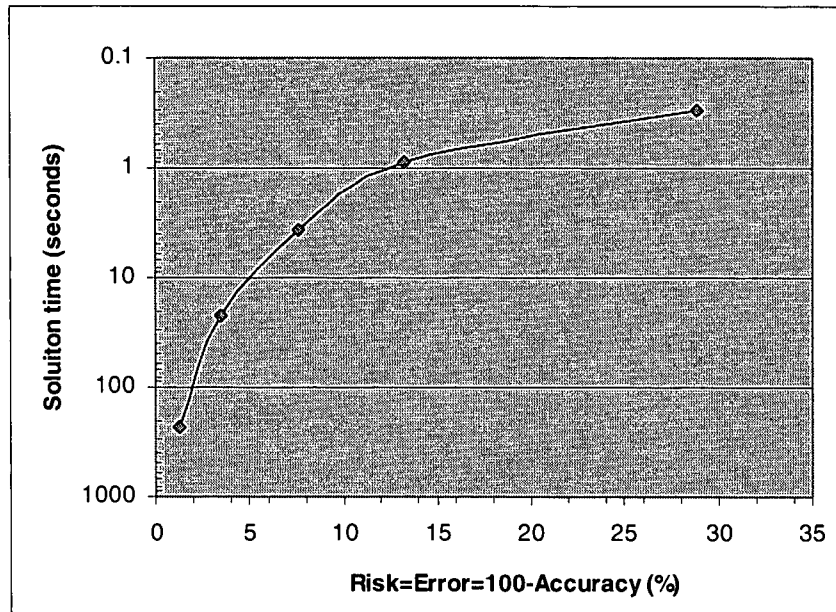


**Figure 2-17: Return versus Risk resulting from changing grid resolution**

The argument of what constitutes risk and what constitutes return is somewhat artificial. It may equally well be said that an inaccuracy constitutes the *risk* and the *return* is how quickly a solution is found. If this approach is used, it is evident from the previous graph that a similar relationship would result if the *x* and *y* axes were interchanged and the following operation were performed:  $Risk = Error = 100\% - Accuracy$ , as increasing error increases the risk of going along an incorrect path. Similarly, the shorter a solution takes to achieve, the higher the return and thus:  $Return = 1/SolutionTime$ . The results of this argument are shown in Figure 2-18 with time on the y axis plotted in reverse order to represent return, *i.e.* a long solution time is



equivalent to a low return, and a short solution time is equivalent to a high return. As previously, the desired relationship between risk and return can be viewed from the curve in Figure 2-18 and a suitable grid spacing can be selected from Figure 2-16. This graph is believed to be more representative of a true *risk-return* relationship than that in Figure 2-17 but its use may be less desirable as the values have to be converted to useable values (*Error* to *accuracy*, and *time* plotted in reverse order).



**Figure 2-18: Modified *Return* versus *Risk* graph for grid resolution**

Taking all of the above arguments into consideration, a grid spacing of 50m is accepted as sufficient for further investigation of other model parameters for this minefield, as it gives almost 97% accuracy. Unless stated otherwise, the figures in subsequent results for this minefield showing path solutions are generated with 50m grid spacing. The nodes are not plotted again in these results for the sake of clarity of the plots.

The analysis of the above minefield gave an interesting insight as to how grid-spacing influences result accuracy. It also determined the setting for the grid spacing to be used in investigation of other parameters for this particular minefield. The results are, however, not useable on other minefields and it is pointless to do such an analysis on every minefield to be investigated because it takes longer than when a single run is done with the finest grid spacing. Thus if path finding on a previously unanalysed minefield is to be done, some

guideline with regard to a generally acceptable selection of grid spacing is required. For this purpose the parameter is investigated in the following paragraph for 20 different minefields, each with a randomly generated minefield of 60 mines.

### 2.5.3 *Grid spacing selection in general terms*

The methodology developed in the preceding paragraph to analyse the effect of grid spacing on result accuracy is applied here to more minefields. This was done in an attempt to generalise the conclusions with regard to the accuracy that can be expected for different grid spacings. For this purpose, 20 random minefields per grid spacing were created via Monte-Carlo simulation. The spacing was again varied by inserting nodes between existing nodes, which in effect halves the grid spacing for each finer setting, again resulting in spacing of 400m, 200m, 100m, 50m and 25m.

The path risk values for the 20 analysed minefields are shown graphically in Table 2-5 and show the general trend of lowering of the path risk value with increasing grid resolution (finer grid spacing). It is further analysed through the mean and standard deviation of the data set associated with each grid spacing. The table also gives an indication of the expected risk value mean by determining the confidence interval half-width with the Student's t-distribution for a  $1-\alpha$  confidence level with

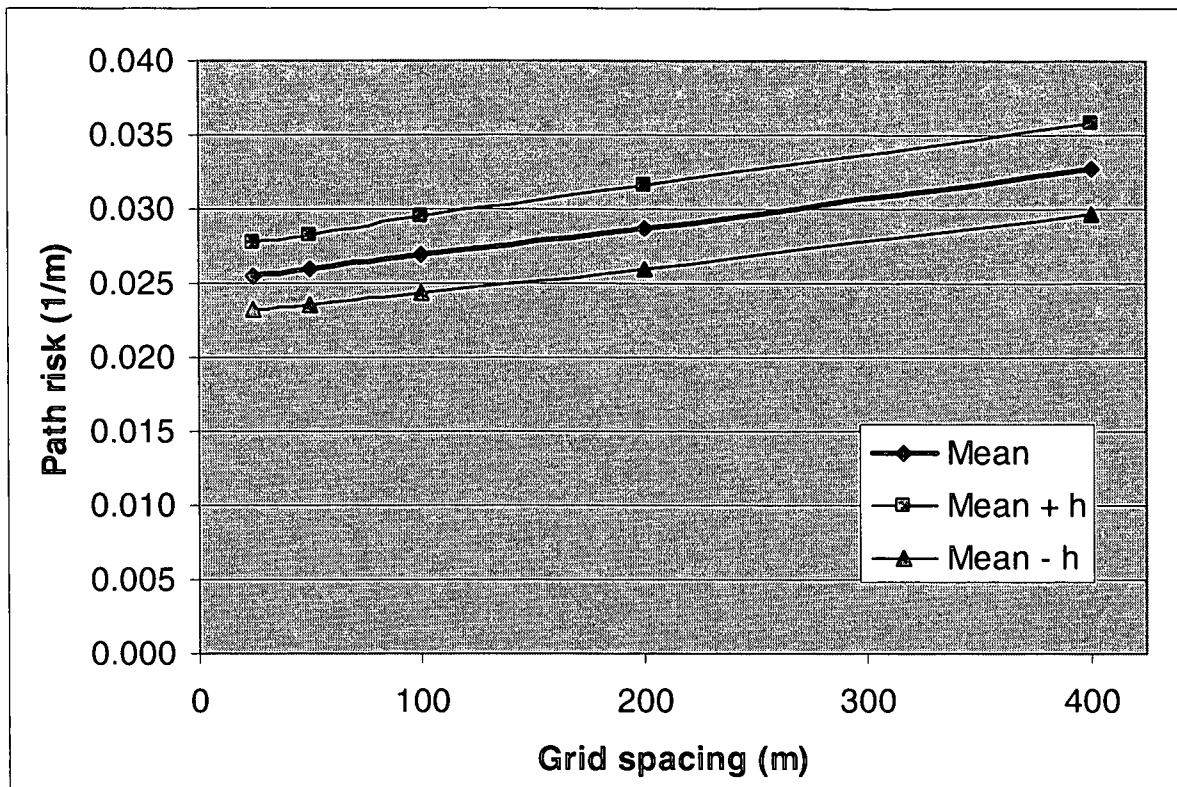
$$h = t_{n-1, 1-\alpha/2} \sqrt{\frac{S^2(n)}{n}} \quad (16)$$

where  $S^2(n)$  is the unbiased point estimate of the variance of the  $n$  observations [20]. The sample size of 20 observations is sufficient to ensure a confidence interval half-width that is within 10% of the expected value. The data mean and confidence interval for path risk is given in Figure 2-19.



**Table 2-5: Path risk values (in  $m^{-1}$ ) and their analysis for 20 randomly generated minefields**

Minefield no (n)	Grid Spacing (m)				
	400	200	100	50	25
1	0.0346	0.0290	0.0270	0.0262	0.0259
2	0.0404	0.0393	0.0380	0.0353	0.0350
3	0.0316	0.0286	0.0250	0.0244	0.0240
4	0.0350	0.0290	0.0268	0.0259	0.0254
5	0.0403	0.0331	0.0308	0.0290	0.0285
6	0.0366	0.0316	0.0302	0.0295	0.0292
7	0.0346	0.0305	0.0259	0.0256	0.0252
8	0.0376	0.0333	0.0302	0.0291	0.0283
9	0.0302	0.0261	0.0248	0.0241	0.0238
10	0.0257	0.0240	0.0236	0.0223	0.0220
11	0.0299	0.0299	0.0280	0.0264	0.0261
12	0.0277	0.0230	0.0225	0.0222	0.0219
13	0.0217	0.0199	0.0193	0.0186	0.0185
14	0.0333	0.0294	0.0284	0.0272	0.0268
15	0.0385	0.0314	0.0294	0.0277	0.0274
16	0.0336	0.0285	0.0275	0.0265	0.0262
17	0.0222	0.0199	0.0189	0.0187	0.0186
18	0.0482	0.0432	0.0399	0.0386	0.0374
19	0.0263	0.0218	0.0207	0.0203	0.0201
20	0.0272	0.0240	0.0222	0.0213	0.0210
Mean $\bar{X}$	0.0327	0.0288	0.0270	0.0259	0.0256
StdDev $S$	0.0066	0.0059	0.0054	0.0050	0.0048
$h_{19,95\%}$	0.0031	0.0028	0.0025	0.0023	0.0023
$\bar{X} + h$	0.0358	0.0315	0.0295	0.0283	0.0278
$\bar{X} - h$	0.0297	0.0260	0.0244	0.0236	0.0233



**Figure 2-19: Path risk confidence interval for 20 randomly generated minefields as a function of grid spacing**

Path risk in general terms, as displayed in the above table and graph, does give the decision maker some idea with regard to expected values and may assist in analysing a specific minefield's safest path in relation to the generally expected values. What is more relevant however is the expected accuracy of a path as a function of the grid spacing. Accuracy was again determined by obtaining an expected path risk value for grid spacing of 0m through regression. This value was then used as the reference to determine accuracy of the path risk values for each grid spacing. This is analysed in Table 2-6 and displayed graphically in the associated Figure 2-20.

**Table 2-6: Path risk accuracy values (in %) and their analysis for 20 randomly generated minefields vs grid spacing**

Minefield no (n)	Grid Spacing (m)				
	400	200	100	50	25
1	64.5	86.2	94.2	97.4	98.5
2	83.3	86.5	90.1	97.9	98.9
3	65.7	78.5	93.7	96.2	98.0
4	60.7	84.3	93.4	96.8	98.6
5	56.4	82.3	90.3	96.8	98.7
6	73.5	90.7	95.5	97.8	99.0
7	60.4	77.1	95.4	96.8	98.4
8	62.9	78.4	89.8	93.8	96.8
9	72.0	89.5	95.1	98.0	99.2
10	82.1	89.8	91.6	97.7	99.2
11	85.1	84.9	92.4	98.4	99.6
12	70.6	92.6	94.8	96.2	97.9
13	82.9	92.5	95.6	99.7	99.9
14	74.3	89.0	92.7	97.2	98.9
15	57.9	84.1	91.3	97.6	98.8
16	71.4	90.9	94.7	98.6	99.5
17	79.7	92.4	97.9	99.1	99.6
18	66.9	80.6	89.7	93.5	96.7
19	67.8	90.4	95.9	97.9	99.0
20	68.7	83.9	92.9	96.9	98.5
Mean $\bar{X}$	70.3	86.2	93.3	97.2	98.7
StdDev S	8.8	5.0	2.3	1.5	0.8
$h_{19,95\%}$	4.1	2.3	1.1	0.7	0.4
$\bar{X} + h$	74.5	88.5	94.4	97.9	99.1
$\bar{X} - h$	66.2	83.9	92.2	96.5	98.3

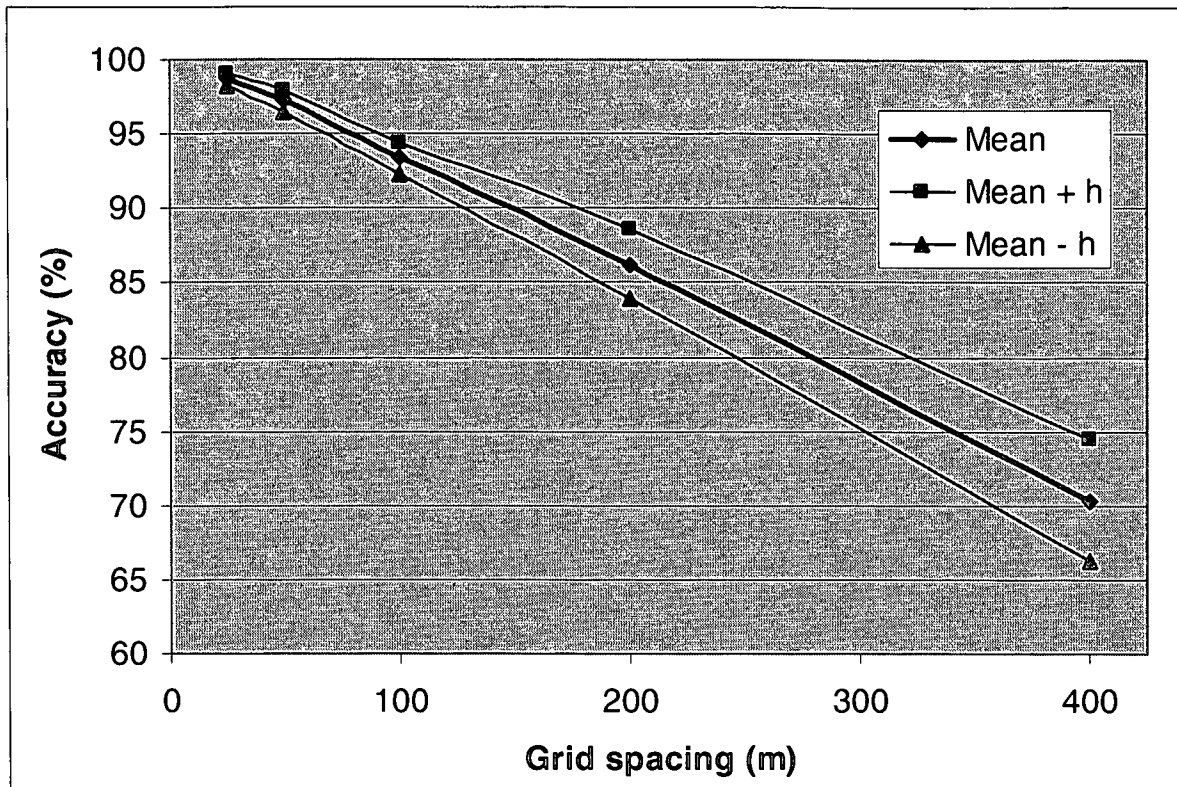


Figure 2-20: Path risk accuracy confidence interval of 20 randomly generated minefields as a function of grid spacing

It can be seen that the expected accuracy improves with increasing grid resolution (finer spacing) and that the previously selected grid spacing of 50m in 2.5.2 (p. 40) would yield an expected accuracy of 97.2% which is 1.5% worse than a 25m grid spacing.

The effect of grid spacing on *path length* can similarly be investigated. It is of interest where optimisation for path length is done such as where a safety radius prevents paths from passing unacceptably closely to mines (refer to § 2.5.9 p.54). It is more difficult to find a generic value as the magnitude of the safety radius is an important parameter. The graph in Figure 2-21 shows the expected accuracy obtained by analysis of 20 randomly generated minefields with a safety radius of 200m which were optimised for shortest path length. This result is relevant to analysis undertaken in CHAPTER 3 (p.62) where a 100m grid spacing was chosen to ensure sufficiently short solution times.



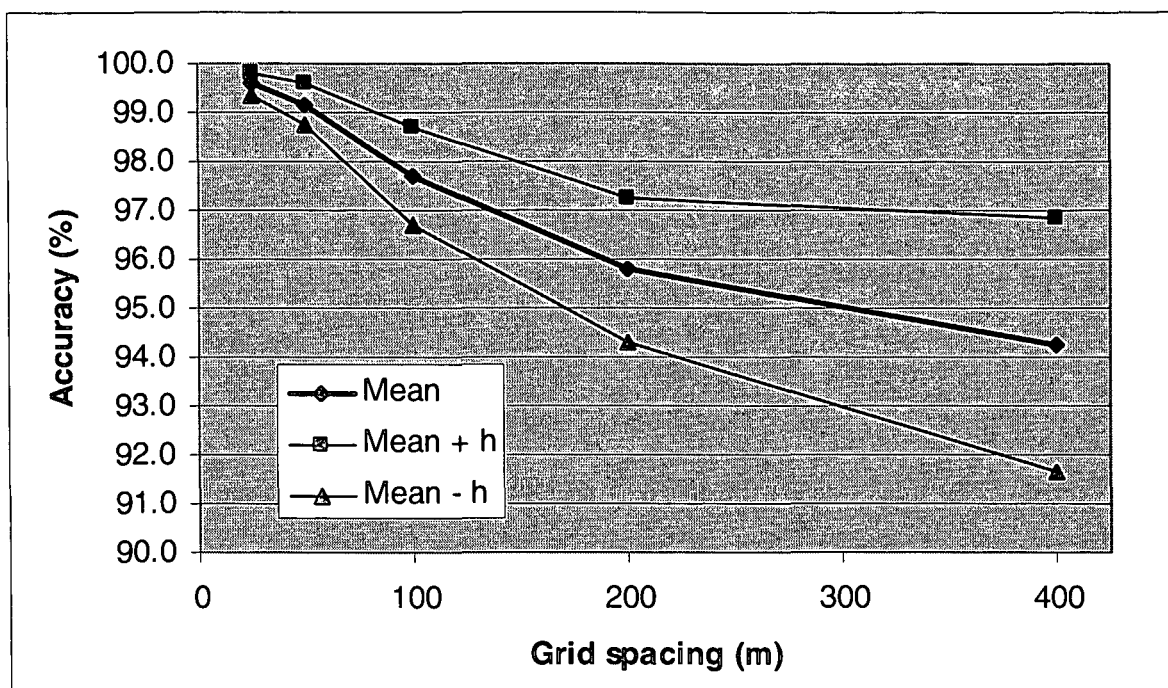
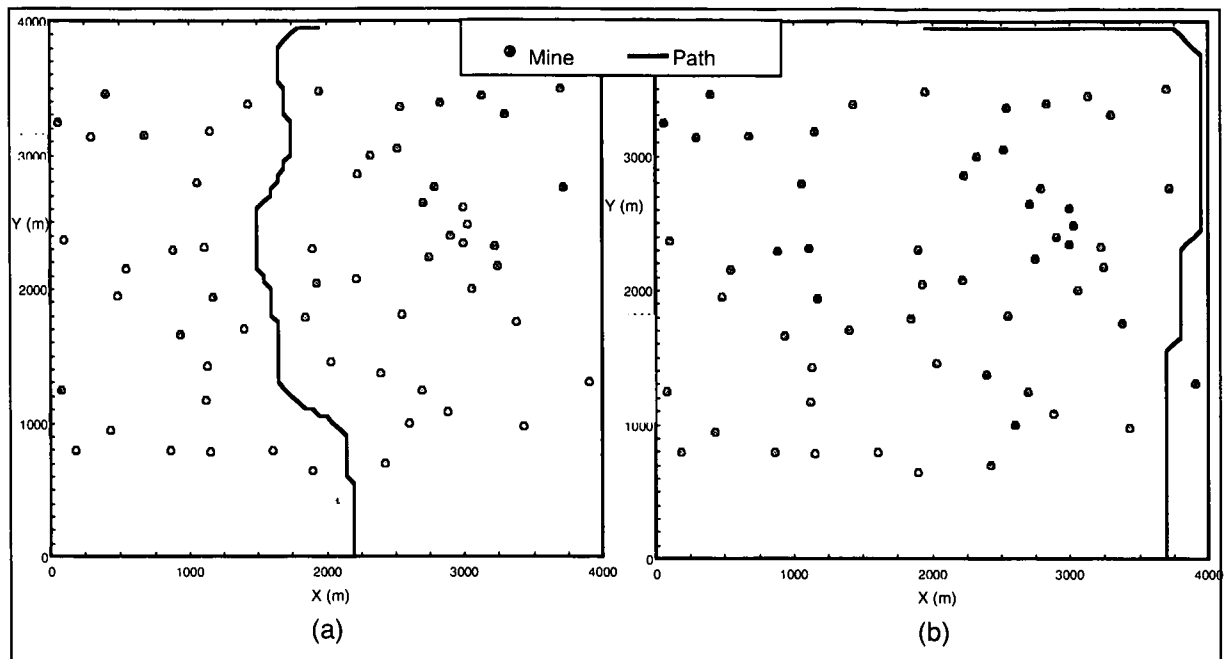


Figure 2-21: Path length accuracy confidence interval for of 20 randomly generated minefields with 200m safety radius as a function of grid spacing

#### 2.5.4 *Paths for ArcRisk = f(closest mine) and ArcRisk = f(all mines)*

Below are the plotted results of the difference in safest path when taking into consideration only the closest mine to the arc as opposed to taking into consideration the cumulative effect of all mines in the minefield. As  $ArcRisk = f(1/R^k)$  it is evident that the effect of mines close to the arc will have a greater effect than those far away. The paths are thus expected to take preference to areas adjacent to sparsely populated vicinities. This effect can be seen clearly in Figure 2-22 (b), where a path is selected that remains as far away from all mines as possible. In contrast to this, Figure 2-22 (a) depicts the safest path when taking into consideration only the closest mine to successive arcs under investigation. The risk setting of including all mines is deemed to be less desirable for general use, as local risk (*i.e.* the closest the path passes any mine) may be increased.



**Figure 2-22: Safest paths generated for arc risk as (a) a function of the closest mine and (b) all mines**

### 2.5.5 Paths for different $k$ values

Shown in Figure 2-23 below are a number of paths showing results for optimisation using different values of  $k$  as defined in equation ( 6 ) (p. 26). It can be noted that the path for  $k=0$  is equivalent to shortest path optimisation as  $ArcRisk=ArcLength$  for any arc. As there is no safety radius the path passes very closely to mines resulting in an unacceptable path.

The results show the magnitude of the effect of  $k$ , although all arc risks are a function of the inverse of distance to a mine. Increasing the value of  $k$  has the effect of forcing the paths further away from the centre of the minefield to areas that are least densely populated, with a subsequent increase in path length. However, for a user, there is in effect only a choice between  $k=1$  representing minimisation of explosion damage if a mine is detonated, and  $k=2$  representing minimisation of mine activation probability.



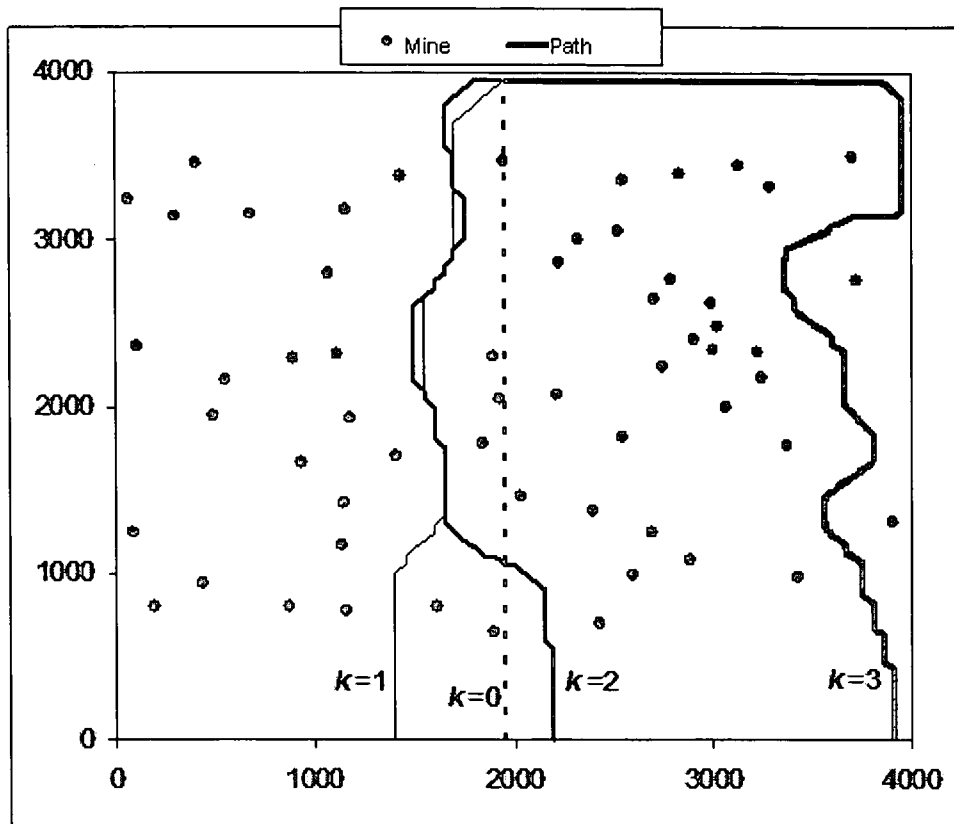
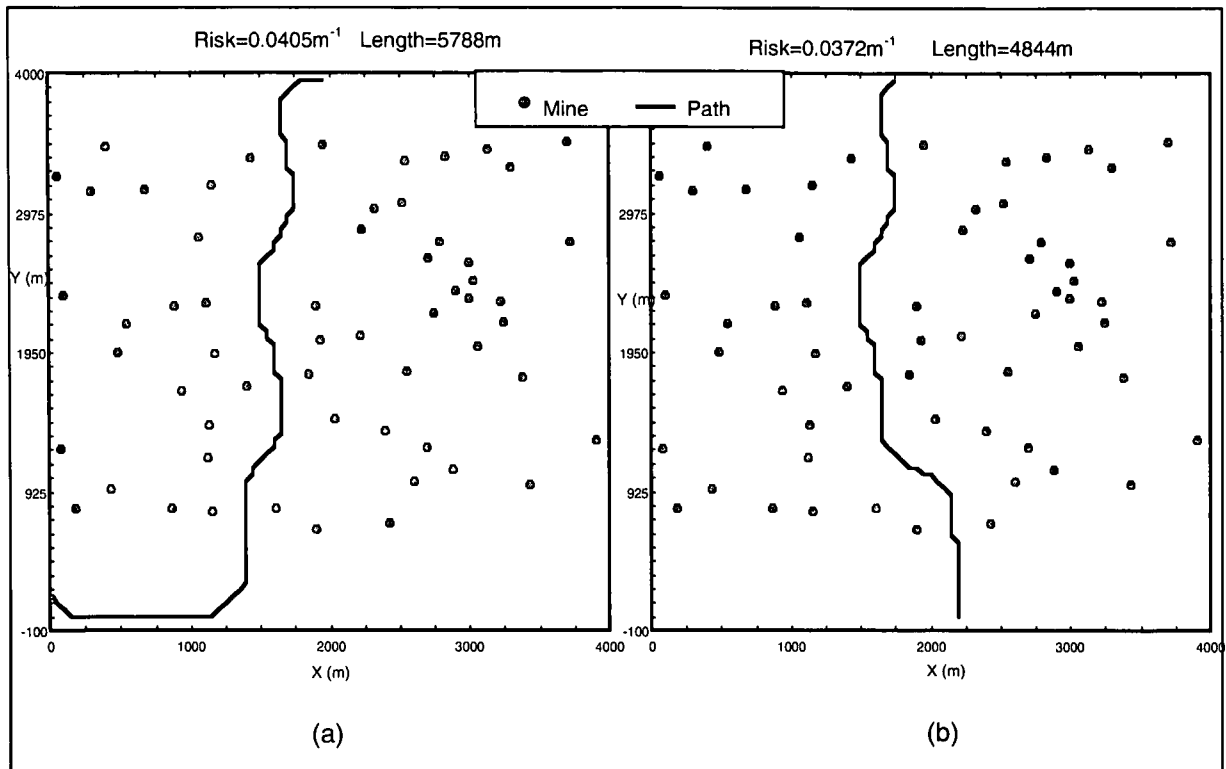


Figure 2-23: Paths generated for different  $k$  values

### 2.5.6 Single and multiple start nodes and end nodes

It is unlikely in normal operational scenarios that a ship will be forced to enter and leave a minefield at an exact start and end position. More likely is the effect (used previously in the results above) where the ship may enter the minefield at any point along the edge of the minefield from which it is approached. It is important enough though to demonstrate the effect of the difference in results when the two approaches are considered.

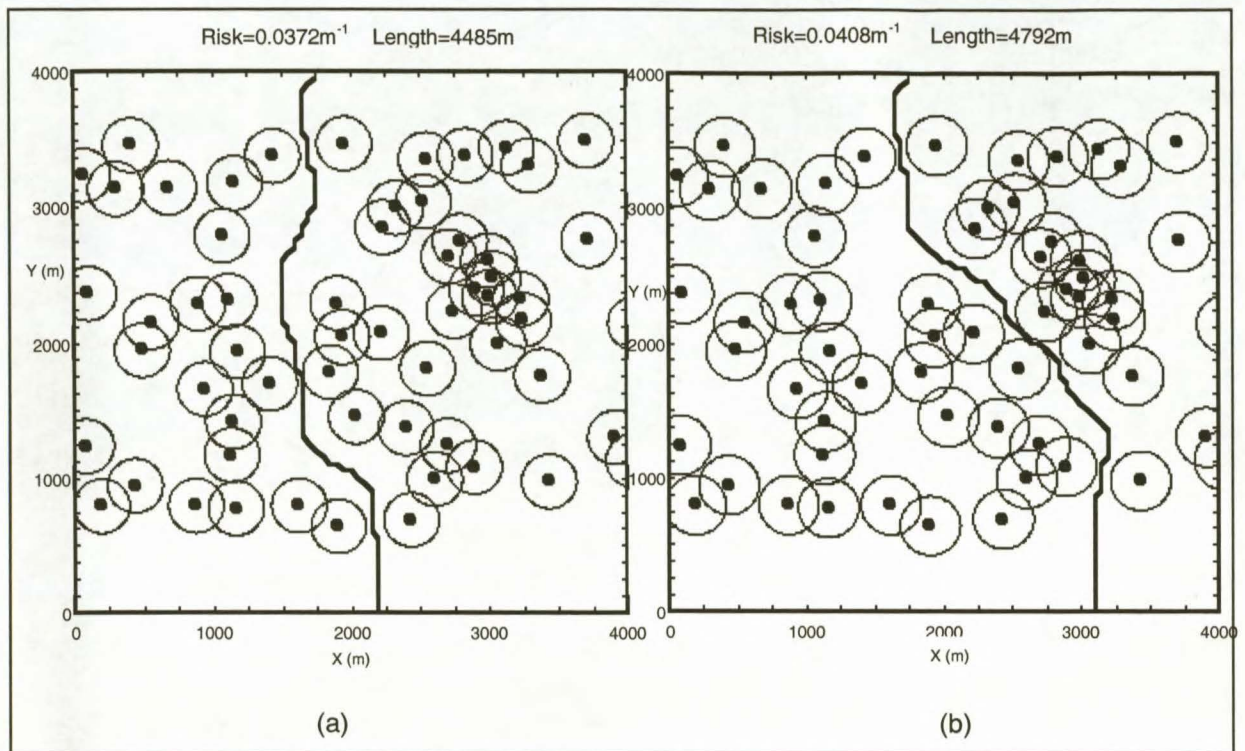
The improvement in *path risk* and *path length* results obtained by introducing multiple start nodes *versus* a fixed start node is shown in Figure 2-24 below. The results are shown for a single starting node (leftmost node in the third row of nodes from the bottom) to the top centre node. Secondly, the result is plotted for the case where starting nodes can be anywhere along the bottom row, and end nodes are the centre 8 nodes of the top row as evident in Figure 2-24 (b). Both path risk and path length are significantly better for the multiple start and end nodes for the given minefield.



**Figure 2-24: Results of path length and risk improvement by allowing multiple start and end nodes**

### 2.5.7 *Safest paths with and without safety radius (specific case)*

The initial implementation attempted to find the safest paths through the minefield with optimisation based on evaluation of total path risk. Such results are shown in the preceding paragraph. The assumption was that there could be instances where the safest path was found, but that it would pass unacceptably close to a mine. To prevent this, the principle of the safety circles around mines is introduced as described in paragraph 2.2.8 (p. 29). Its effect on path length and total path risk value is illustrated in Figure 2-25, where it is shown how an increasing safety radius has modified the safest path that can be found. It is clear that a slight further increase in safety radius may close down all paths resulting in no valid path being found. The effect on the initial path is small, with an increasing safety radius as the routing algorithm forces the path to pass equidistantly to mines. This means that the path will only change once the safety circles of two adjacent mines, between which the path passes, overlap.



**Figure 2-25: Path change with safety radius increase from (a) 200m to (b) 220m**

### 2.5.8 *Safest paths with and without safety radius (general trends)*

On evaluation of a number of different random minefields where the safety distance was increased, it was found that in general the safest path found would remain unchanged until the safety distance increased to such a level that one of two effects were observed:

- The path made a sudden increase to an almost entirely new path with a large increase in the risk value of the path.
- The circles intersected to such a level that no unobstructed path was available any longer, and no solution was found.

It was deemed necessary to establish and, if possible, to quantify to what degree the safety circle influenced safe paths. This was done by running simulations for different randomly generated minefields. For each new minefield the safety circle was incremented from 0m to 400m in 20m increments. The expected output would be a distribution of when the safety circle **changed the path risk for the first time** (indicated by a change in path risk value) and when it resulted in **no path being found**. The results are displayed in Table 2-7.

**Table 2-7: Magnitude of safety radius in order to affect path optimisation**

Minefield no.	Path change effected	No path possible
1	280m	280m
2	280m	280m
3	300m	320m
4	260m	280m
5	380m	380m
6	260m	260m
7	300m	300m
8	260m	300m
9	360m	360m
10	260m	260m
11	340m	360m
12	300m	320m
13	220m	300m
14	240m	260m
15	220m	240m
16	220m	300m
17	340m	340m
18	260m	280m
19	240m	280m
20	280m	280m
<b>Maximum</b>	<b>380m</b>	<b>380m</b>
<b>Minimum</b>	<b>220m</b>	<b>240m</b>
<b>Mean</b>	<b>280m</b>	<b>299m</b>
<b>Standard Deviation</b>	<b>46.34m</b>	<b>37.54m</b>

If the mines were evenly distributed over the equivalent area as in the simulation, there would be one mine to cover  $203000\text{m}^2$  ( $2900\text{m} \times 4200\text{m}/60$  mines). This implies that a mine would fall into a square of approximately  $450\text{m} \times 450\text{m}$ . It is evident that it would require a safety radius of roughly  $450\text{m}/2=225\text{m}$  to ensure that the safety circles of adjacent mines touched so that no path is possible. This is the smallest safety radius with which it would be possible to close off a minefield for penetration for an *evenly* distributed minefield. The random generation of the examined minefields causes an unevenly spaced mines through which the optimisation finds the path via the least dense areas. It therefore seems correct that the measured safety radius required to close a minefield should be somewhat larger in order to close off areas less dense than an average density. The mean measured safety radius is a factor of  $299/225=1.33$  larger than the theoretical minimum. It is of interest to minefield designers to establish ratios for other densities, but it falls outside the scope of this thesis as it is not

relevant to the optimisation processes described here. It has, however, been shown that this methodology can be used as a tool to establish the ratio.

It is concluded that the introduction of the safety radius is an important aspect in ensuring safety to transiting ships. As such it ought to be introduced as standard practice to remove local risk. It is evident from Table 2-7 that the mean safety distances for path modification (making a path safer by removing a local risk even at a the cost of a higher path risk figure) and path closure (no more valid path through the minefield) are mostly quite close to each other (280m *versus* 299m). Therefore, in general a safety distance will either have no effect on the path if small enough, or result in path closure if chosen large enough. The fact that the safety radius has a relatively small effect in general on *safest paths* is an indication of the efficiency and correctness of the method of optimising with regard to *risk*. The safety radius does, however, provide the opportunity to optimise a path for *length*, provided a minimum acceptable distance to any mine is ensured by implementing a safety radius. This is discussed in the next paragraph (2.5.9).

#### 2.5.9 Shortest paths with changing safety radius

The introduction of the concept of a safety radius has introduced another risk parameter into the optimisation exercise. It could be argued that under specific operational conditions (such as when time is of importance) it may be necessary to find the shortest path through the minefield, provided a minimum risk value is not exceeded. This minimum risk figure can be locally applied through the safety radius as it ensures that the ship does not come closer to a mine than the desired minimum distance at any point.

This implementation requires that the risk cost function allocated to an arc and used for path optimisation (described in paragraph 2.2.7, p. 23) is now replaced by a traditional arc distance. The optimised paths are now the ones giving the *shortest path* through the minefield. It is important that arc risk still remains available, as it will determine the total risk figure for the path. Previously, paths were optimised for risk, but the path length was a secondary characteristic. Now the paths are optimised for distance and path risk is the secondary metric. Again it was of importance to investigate the effect that increasing safety distance has on path length and path risk figure. The important parameters are shown in the



figures below for a random minefield selected to be representative of a case where the increase in path length is relatively smooth with increasing safety radius. Because of the time required for a complete run, only 5 minefields were evaluated and although all minefields show the same general trends, this one was chosen, as the data series are evenly distributed. The data was generated by increasing safety radius by 10m increments from 0m until no more paths could be found. The effect of increasing the safety radius can be seen in the graphs in Figure 2-26.

In Figure 2-26(a) it can be noted that there is a stepwise increase in some instances, which can be ascribed to the fact that there are different paths with the same distance through the minefield. These paths do not have the same risk values as can be seen when comparing the shape of the data points of the graphs in Figure 2-26 (a) and (b).

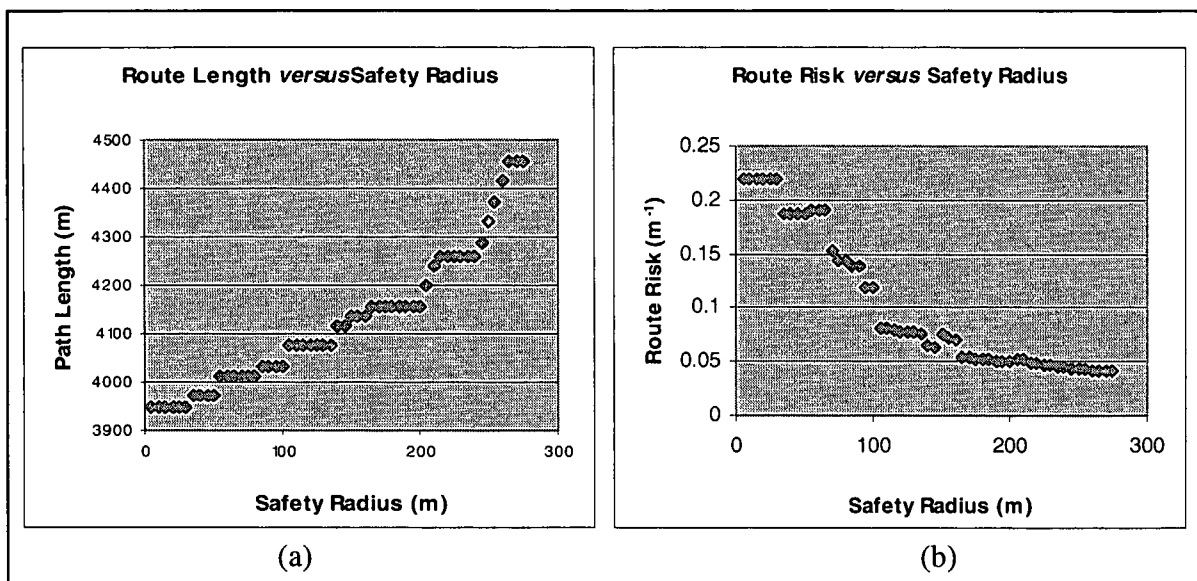
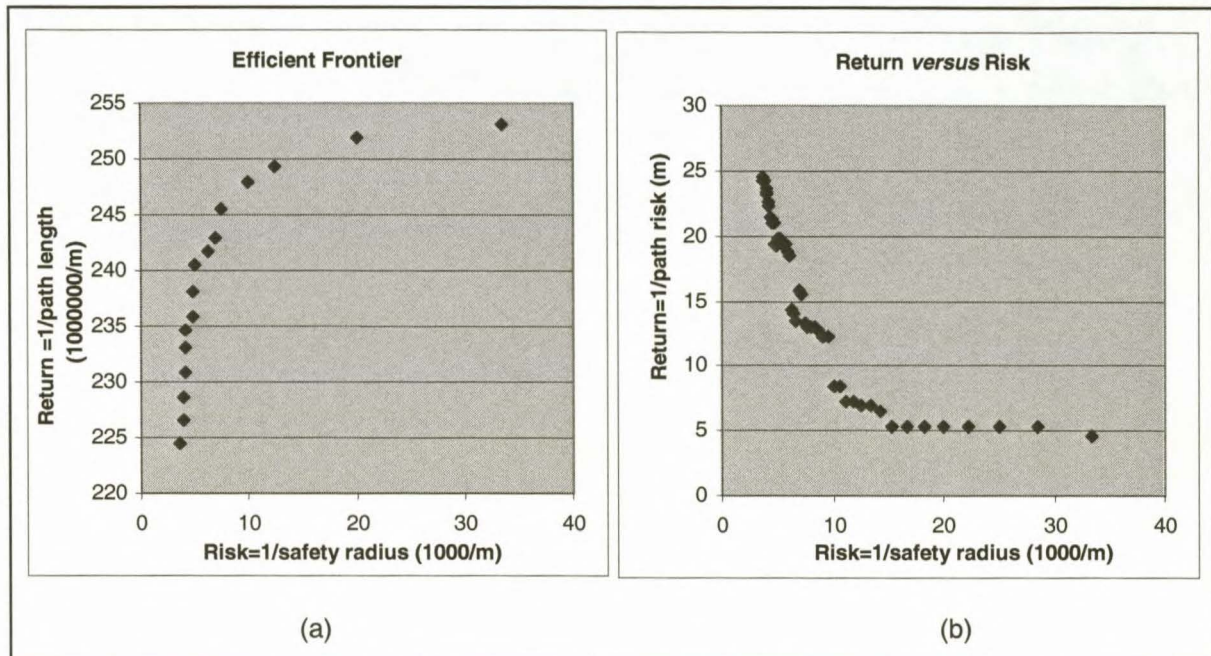


Figure 2-26: Change of path risk and path length with increasing safety radius

The graphs can be converted into risk-return graphs (Figure 2-27) by inverting the  $x$ -axis to obtain increasing risk from decreasing safety distance. Similarly the  $y$ -axes have to be inverted to obtain increasing return. For path length, the shorter the path, the higher the return. Similarly for path risk, the lower the risk, the higher the return. As the *efficient frontier* is the collection of efficient solutions (no higher return possible for the same risk), those paths that have a longer path for equal risk (safety distance) are excluded from the plots.

Figure 2-27(a) can be viewed as a true Efficient Frontier Graph, as it represents the best obtainable return (shortest path obtained by shortest path optimisation) for a chosen risk (safety radius). The associated graph Figure 2-27(b) representing return based on path risk, is not to be viewed as an Efficient Frontier Graph, as the data points have not been obtained by optimising for risk. They are merely an associated value of the path risk.



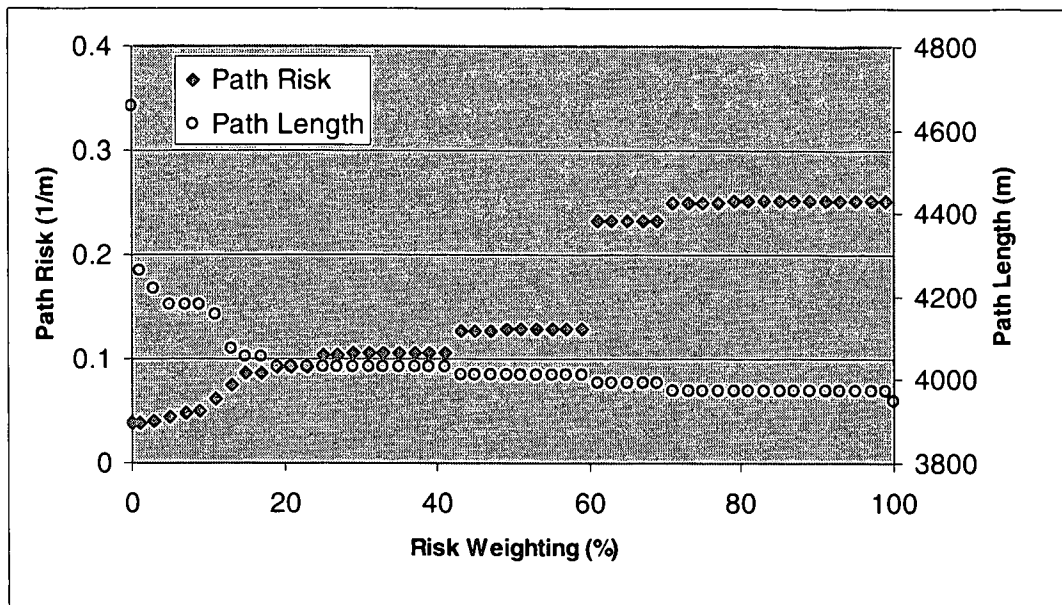
**Figure 2-27: Return versus risk for increasing safety radius**

As the changing safety radius generated a number of paths through a minefield, it was deemed relevant to evaluate the relationship between the risk value and the path length of such different paths. To create such an Efficient Frontier Graph, it was necessary to find the efficient solutions to be plotted. This means that for each return ( $1/\text{path length}$  as in Figure 2-27 (a)) it had to be ensured that there was no path with a lower risk. This can only be achieved if the objective function optimises for path length *and* path risk.

It had thus become necessary to create a function for arc cost that included distance and risk values. It would be ideal if the ratio or weighting of the two components could be varied according to the decision maker's requirement. Such an implementation is presented in the next paragraph (2.5.10).

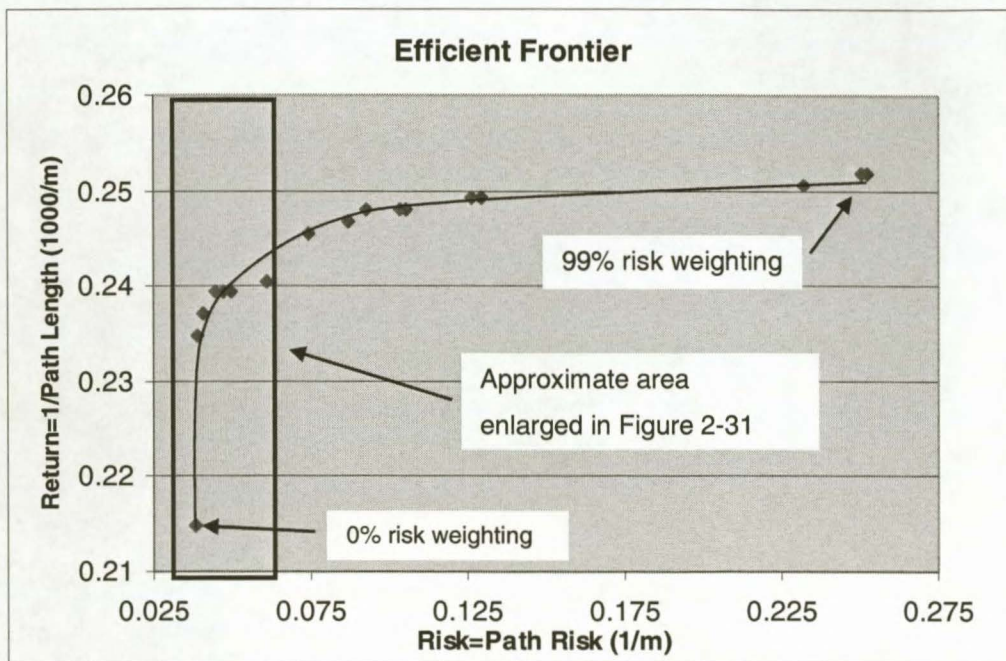






**Figure 2-29: Path risk and path length versus risk weighting**

The risk (*Path Risk*) and return ( $1/\text{Path Length}$ ) for the above runs were subsequently plotted against each other in Figure 2-30 to form an Efficient Frontier Graph representing return *versus* risk for the given minefield. The data was not plotted for a 100% risk setting (shortest path) to display the effect that even minimal inclusion of risk in the objective function has on the path risk value. Because a 100% risk setting chooses the shortest path, for the case where there is no safety radius, it is a straight vertical line to the end node. This causes the path to pass very closely to mines and thus results in a very high risk value (see path in Figure 2-23 p. 50 where  $k=0$ ). The path risk value for such a path through the featured minefield is  $2.5\text{m}^{-1}$  and as can be seen would distort the plotted results in Figure 2-30, where the maximum plotted risk value is approximately  $0.25\text{m}^{-1}$ . This implies that for an increase of 1% risk weighting (from 99% to 100%) the path risk value increases by a factor of 10. Similarly, in the case where there is an increase from 0% to approximately 20%, there is a large reduction in path length for a minimal increase in path risk (refer Figure 2-28). This area is deemed of special interest and is enlarged from the Efficient Frontier in Figure 2-30 and shown in Figure 2-31. The line superimposed on the plots in Figure 2-30 and Figure 2-31 is added to show the general trend. It is not to be interpreted as the efficient frontier.



**Figure 2-30: Efficient Frontier Graph for the selected minefield, generated by varying the risk weighting**

Figure 2-31 indicates that the Efficient Frontier behaviour also applies to the area at the start of the graph shown in Figure 2-30. A small sacrifice in terms of risk gives a good improvement in path length, whereafter a further sacrifice in path risk does not improve the path length at all. This information is not clearly evident from the graph displaying the full risk return relationship. The conclusion can be drawn that if optimising for risk, there is a need to investigate the low risk weighting area closely to determine whether a risk sacrifice is desirable in order to obtain a path that is lightly optimised for path length as well.



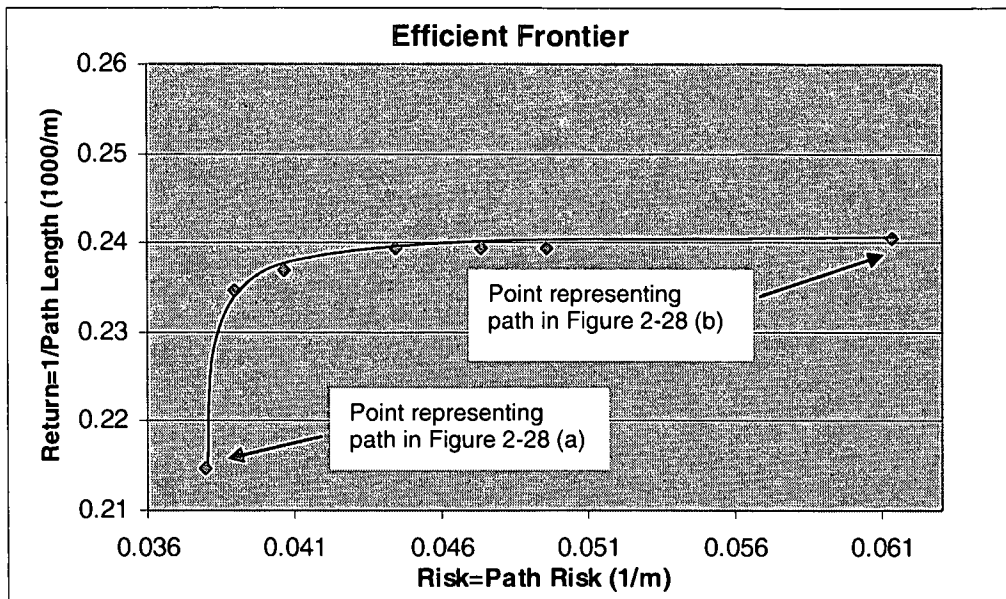


Figure 2-31: Enlarged area from Figure 2-30, showing Efficient Frontier

The above investigation shows that a pure optimisation for *path length* or *path risk* may not always be desirable. It may be acceptable or even preferable to include a small percentage of optimisation for the other parameter so that a balanced path results. This has to be done for each individual minefield by investigating the full relationship of path risk *versus* length, as was done here. The Efficient Frontier Graph will determine how desirable a sacrifice may be, and what a good choice of risk weighting may be for that minefield.

## 2.6 Conclusions on path optimisation investigation

In this chapter it was demonstrated how a method can be developed to enable risk management in transiting a charted minefield by using Dijkstra's Algorithm as foundation. The quantification of risk is based on parameters that relate to probability of detonating a mine or alternatively sustaining damage from an exploding mine together with the resulting path length. The important user selectable settings were investigated through selected examples and their effect was graphically presented.

It was shown that risk can be managed by either optimising paths with regard to the total risk value for a path, made up of all the path's arc risks. Alternatively, the maximum allowable risk could be selected (through a safety radius) and the path could be optimised for distance.

In both instances it is advisable to include a risk weighting that combines some percentage of the parameter that is not primarily being optimised for. Thus, when optimising for path risk, a small percentage of optimisation for distance should be included. This can result in a much improved path length while sacrificing very little in path risk. Alternatively, when optimising for distance, the inclusion of a small percentage of optimisation for risk can result in much safer paths with very little sacrifice of path length. The desired amount of risk weighting has to be established through a risk-return visualisation such as an Efficient Frontier Graph.

The examples shown here are all minefields through which a path has been possible by virtue of the selected risk settings. However, there is an operational possibility that no path with an acceptably low risk-profile exists. The method discussed in this chapter provides no solution to such a situation. Operationally, one strategy that can be used to overcome such a problem is to remove mines until a sufficiently safe path results, when evaluated by the methods described in this chapter. In the next chapter (CHAPTER 3), a method is proposed which can be implemented to enable removal of the impeding mines to achieve transit through an otherwise impenetrable minefield.

### CHAPTER 3 MINE REMOVAL OPTIMISATION

In the previous chapter all the examples that were presented were such, that paths were possible through a minefield. It is, however, easy to imagine that the introduction of a large enough safety radius could create situations where mines are spaced in a way that the resulting safety circles intersect so that no safe path can be found. An alternative situation could be defined where in the absence of a safety radius, the total path risk of the best path is considered to be too high. Both these scenarios would require the removal of one or more mines to ensure that a viable path results.

In this chapter the methodology with which to solve the second part of the stated problem of finding a sufficiently safe path through a minefield is presented. The theoretical background required for the solution is given, whereafter the implementation is discussed, both in creation of a software based tool and in presentation of sample results.

The removal of the mines could be considered to be optimised if the minimum necessary mines are removed in order to enable a sufficiently safe path. To achieve this is a typical case of a combinatorial optimisation problem. If only a few mines need to be removed, the most effective solution would be an exhaustive search, *i.e.* all combinations are investigated. If removal of one mine would be sufficient to enable a safe path, the number of tests for the best path would be equal to the number of mines. If, as in the previous examples, the minefield consists of 60 mines, the mines can be removed one at a time and the path optimisation described in the previous chapter can be applied. This implies 60 runs each time with one mine removed. Some of these runs will not result in a safely passable minefield. If, out of the 60 trials, more than one safe minefield results, the solution is selected that provides the shortest path or the safest path, depending on the preference of the user. However, if the 60 trials do not result in an acceptable solution, and two mines need to be removed, the number of trials to find the best path escalates up to  $\binom{60}{2}=1770$ . The required number of trials increases rapidly, with removal of 6 mines requiring  $\binom{60}{6}\approx 5\times 10^7$  investigations. If it can be optimistically estimated that each investigation takes 1 second to perform (path optimisation as previously discussed required approximately 20 seconds at the required resolution) then it

would take approximately 1.5 years to find the best path. For a minefield with 80 mines (as investigated in the results that follow below), the solution time increases to approximately 9.5 years if 6 mines have to be removed. This is clearly not a feasible solution approach and a more efficient method is required.

Below a sub-optimal method is described with which selection of mines to be removed can be satisfactorily achieved. Hence optimised results can be expected, although an optimal solution is not guaranteed. The proposed solution is dealt with in the following logically structured way:

- description of the problem by finding similarities and differences to existing defined problem types;
- investigation of alternative approaches and suggestion of solution methodology;
- description of theoretical background required to implement the proposed solution;
- description of computer implementation of the solution;
- implementation of improvements;
- presentation and discussion of results of the implemented solution.

As the problem is one of combinatorial optimisation, this had to be taken as the starting point in searching for a solution to the mine removal problem. Many solutions for combinatorial optimisation exist, of which some are highly efficient for a suitable problem type such as Dijkstra's Algorithm described before. The class of optimisation required here is entirely different and when compared to existing types of combinatorial optimisation problems, may be related to the *Knapsack Problem* which is described and analysed for similarities below.

### 3.1 The Knapsack Problem

This is a classical combinatorial optimisation problem and has been thoroughly analysed for both exact and heuristic (sub-optimal) solutions. The problem can be described informally as having to optimise items (each with a utility value and a penalty) by selecting an optimum combination of such items to fit into a constrained capacity so that the highest utility is achieved [21],[22].

Mathematically the problem is described as consisting of  $n$  items and a knapsack with capacity  $C$ . Each item has a weight  $w_i$  and a profit  $p_i$ . The knapsack needs to be filled with

items to achieve maximum profit without exceeding its capacity. This means a binary assignment vector  $\vec{x}=(x_1,x_2,\dots,x_n)$  has to be found where  $x_i \in \{0,1\}$ , such that  $\sum_{i=1}^n w_i x_i \leq C$  and for which the profit  $P(\vec{x})=\sum_{i=1}^n p_i x_i$  is a maximum [23].

Although not identical in nature to the problem of mine removal that has to be solved here, there is the parallel in that the best selection has to be made from a number of available objects. The utility or value of the objects (here the mines) is related as to how they assist in reducing path risk. But instead of their weight having to be considered to ensure the objects do not exceed a maximum, their weight (number of mines) has to be minimised. This minimisation takes preference over the utility value, as the minimum number of mines have to be removed rather than the safest path being created.

The exact parallel to the Knapsack Problem would be that a certain amount of resources are available (the constrained knapsack) to remove the mines. This could be the time or the resources used to remove mines. Given this constraint, the best mines which can be removed with the available capacity have to be selected so that the safest path results.

Nevertheless, the stated problem was seen to be similar enough to the Knapsack Problem so that solutions which were suited to the Knapsack Problem could be considered to solve the mine removal problem. Solutions that were reported as being available to solve the problem were listed as linear programming, integer programming and Genetic Algorithms. Because of its described suitability for combinatorial optimisation, it was decided to implement a solution using a Genetic Algorithm. A description of Genetic Algorithms is given in the next paragraph.

### 3.2 Genetic Algorithms

The Genetic Algorithm (GA) is an optimisation approach that mimics the principle of “survival of the fittest” found in nature. Solutions (also called chromosomes) that are generated are not guaranteed to be the best, but if correctly applied can provide sufficiently good solutions. A number of possible solutions (called a population) to a problem are represented in a way reflecting genetic makeup. The best solutions survive and can be modified by exchanging “genetic material” from other good solutions (crossover) or by



randomly modifying parts of the genetic structure (mutation). Subsequent generations (children or offspring), created by merging genetic material from two good previous solutions (parents), improve steadily until convergence is achieved. The first step in applying a GA is to select a *genetic coding method* (makeup) that represents a suitable chromosome. The second important aspect in GA optimisation is the *fitness function* with which a solution is evaluated and ranked relative to other solutions. Both of these have to be selected to suit the specific problem type. The pseudo code for a GA is shown in Figure 3-1 (for detailed discussions of the GA, see *e.g.* [24]).

```

Set Generation ← 0
Generate a random initial population
Repeat
  If the crossover probability is satisfied Then
    Select two different chromosomes randomly from the population
    Of the two selected above, set the chromosome with the best fitness function to be the first
    parent
    Choose a random chromosome from the population
    Let the chromosome chosen in the step above be the second parent
    Perform crossover on the two parents but generate only one child/offspring
  Else
    Choose any of the two parents to be the offspring
  End If

  For Each gene in the offspring chromosome
    If mutation probability is satisfied Then flip the value of the gene
  End For Each

  Evaluate the offspring chromosome with the fitness function, recording the result
  If the offspring is better than the least fit chromosome in the population Then
    Replace a random chromosome in the population with the offspring chromosome
  End If
  Increment Generation
Until stopping condition is true

```

**Figure 3-1: Pseudo code of a Genetic Algorithm**

### 3.3 Genetic Algorithm solution implementation

As the *coding* of the genetic makeup has to be matched to the problem definition, it had to be addressed first. The most common type of coding found was that using a binary value for an item in a Knapsack being present or absent, an example of which can be found in [21]. In this application, suppose a mapped sea minefield contains  $n$  mines, with the mines scattered in this field. The genetic coding is implemented by creating a chromosome consisting of a string of genes equal to the number of mines, each mine number being represented by its position in the string. If mine 5 is to be removed, the value of the gene 5 is 0; if the mine is to remain

present, the associated gene's value is 1. The drawback here was that relatively long strings resulted, their length being equivalent to the number of mines in the minefield. Moreover such an encoding is fairly intuitive and easy to implement.

The *fitness figure* is a function of the number of mines that have been removed (fewer is better) and the resulting path length, *i.e.* a shorter path length is better. This implies that every member of the population has to be evaluated for its path length with the shortest path method described in CHAPTER 2. The fitness function is defined as follows:

$$f(L,D) = \frac{2}{\frac{L}{L_b} + \frac{D}{D_m}} \quad (17)$$

Where

$L$  = path length resulting from a chromosome structure

$L_b$  = shortest possible path through the minefield in absence of mines

$D$  = number of mines to be removed for a chromosome

$D_m$  = Minimum number of mines that can be removed

The best path  $L_b$  is defined as the shortest distance between the departure point of the ship and its destination, *if there are no sea mines present*. The value of  $D_m$  is equal to one sea mine, since that is the minimum number of mines to be removed to obtain a safe path. If zero sea mines must be removed, then the problem can immediately be addressed with the strategy proposed in CHAPTER 2. It follows that the possible values resulting from the fitness function are limited to the range [0;1] and these values are dimensionless. Note that only the shortest path with acceptable risk approach was considered here, while the minimum risk approach (see CHAPTER 2) could also be followed. The safest path approach, however, requires a greater computational effort while the GA is executing, and because of the highly iterative nature of the GA, it was not implemented. Instead it was accepted that the risk reducing method of safety radius and shortest path would be sufficient to prove the viability of the method described here. The algorithm developed for the analysis described in CHAPTER 2 was adjusted so that risk needed not be calculated for arcs. Instead the arc length only was used for the cost function of the arc, and was optimised. This improved the execution time immensely. The improvement was not quantified exactly but was measured at approximately 30% reduction for a similar model execution (11.7 seconds with path risk

method *versus* 7.9 seconds with shortest path only). A further execution time improvement was achieved by increasing grid spacing to 100m (with an expected 93% accuracy as determined in Table 2-6). This reduced solution time for one iteration to 0.1 seconds and made it useable as a fitness evaluation tool in the GA.

For initialisation, a random population is generated with an assignable number of mines being randomly removed from the minefield for each specific population member. Each member of the population is measured against the fitness value resulting from the shortest path optimisation and number of mines removed. The best solutions continue to reproduce for a number of iterations (generations). Allowance was made in the software developed for this study for a minefield containing up to 100 mines, but in the results below a minefield containing 80 mines is presented. A safety radius of 200m was used unless indicated otherwise.

Subsequent generations are created by exchange of genetic material between good solutions from the previous generation. This can be achieved in different ways and the options implemented here and investigated in the next paragraph are:

- Criteria for qualification as being a member of the new generation:
  - The one option is to sort the old and the new generation from good to bad and select the top performing members (number = population size) to be the new generation. This ensures that the best old solutions remain within the population and can be available as parents for the next generation.
  - The second option is to keep only the new generation and discard the previous generation. This introduces the maximum new genetic material into a generation but may prevent a sufficient number of the good old solutions from surviving.
- Criteria for being selected as a parent for the new generation:
  - **Roulette wheel selection:** (for details see p. 237 in [24]). Here the first parent is chosen at random in such a way that the better a solution's fitness, the better the chance of being selected (hence the choice is weighted according to the fitness distribution). The second parent is selected by

randomly choosing from the rest of the population, each having an equal chance of being selected

- **Best mates all:** Here the solution with the highest fitness is taken as the first parent and the second parent is itself and each of the remaining members of the current population. (This is not deemed to be a good strategy as it is almost impossible to move away from a local optimum. However, it may be of value for routing as used here, as the first and/or last sections of paths may be preserved only and the centre parts made up of new solutions.) The best solution of a generation is ensured of surviving unchanged to the next generation, unless it is changed by mutation.

### 3.4 Random search algorithm

During development and early application of the GA, the idea was formed to obtain a better way to initiate the starting population in place of the random deletion of a percentage of genes representing mines. It was attempted to form members of the initial population that had complete paths (*i.e.* paths that led from start node to end node) by removing only the least number of mines to enable such a path. The aim of this was to create feasible path sections which could be combined in different combinations by exchange of genetic material in subsequent generations. The length of the path was unimportant for the first generation.

The method to obtain these “random” paths was implemented by starting with a minefield containing no mines. Mines would be randomly chosen to be placed back in their positions in the minefield one at a time. After each replacement, the path optimisation would be performed to check if a valid path still existed. If a path was not possible, the last placed mine would be removed again and kept out of the possible replacements. This was done until all the mines of the original minefield were tested for replacement. This “valid” combination of mines would then become a member of the initial population. The results of this process are discussed in 3.5.4 (p. 75). The new methodology was again implemented in Extend<sup>®</sup> and the graphic interface to the user is shown in Figure 3-2 while model detail and detailed user interface are shown in Figure 3-3. The computer code in ModL for the GA optimiser block including the Random Search Algorithm is given in APPENDIX B.

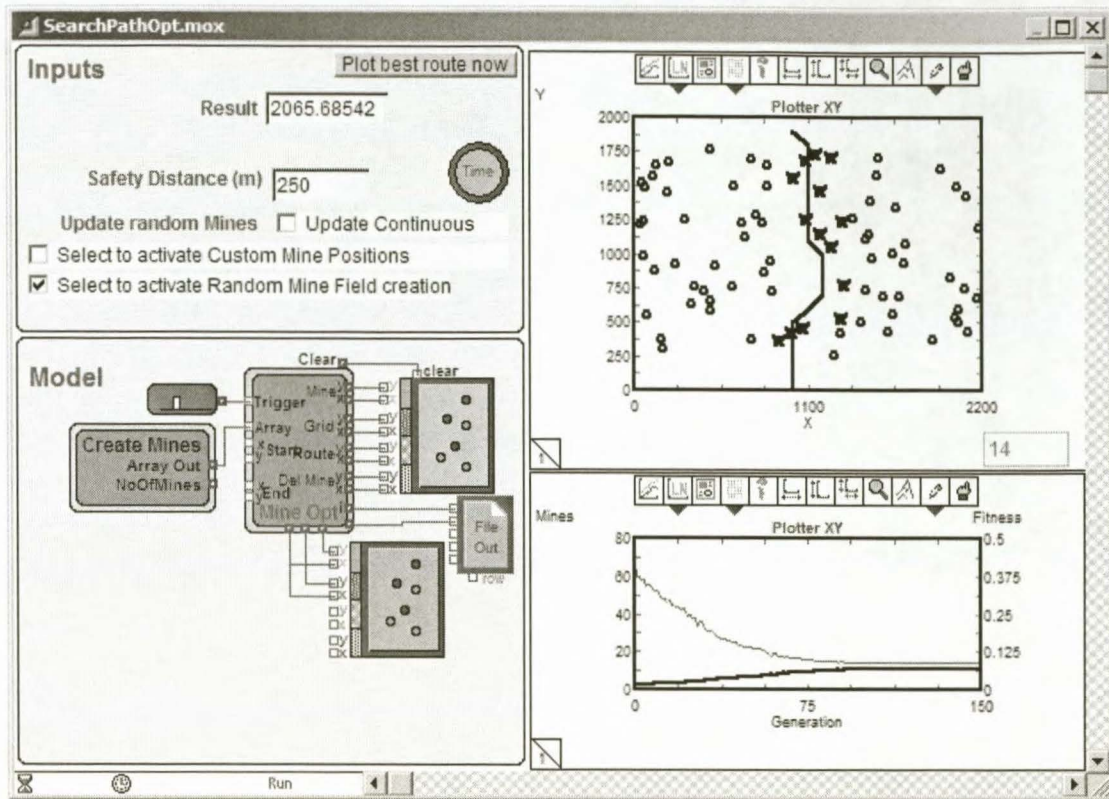


Figure 3-2: Extend<sup>®</sup> genetic algorithm optimisation model with reduced user interface

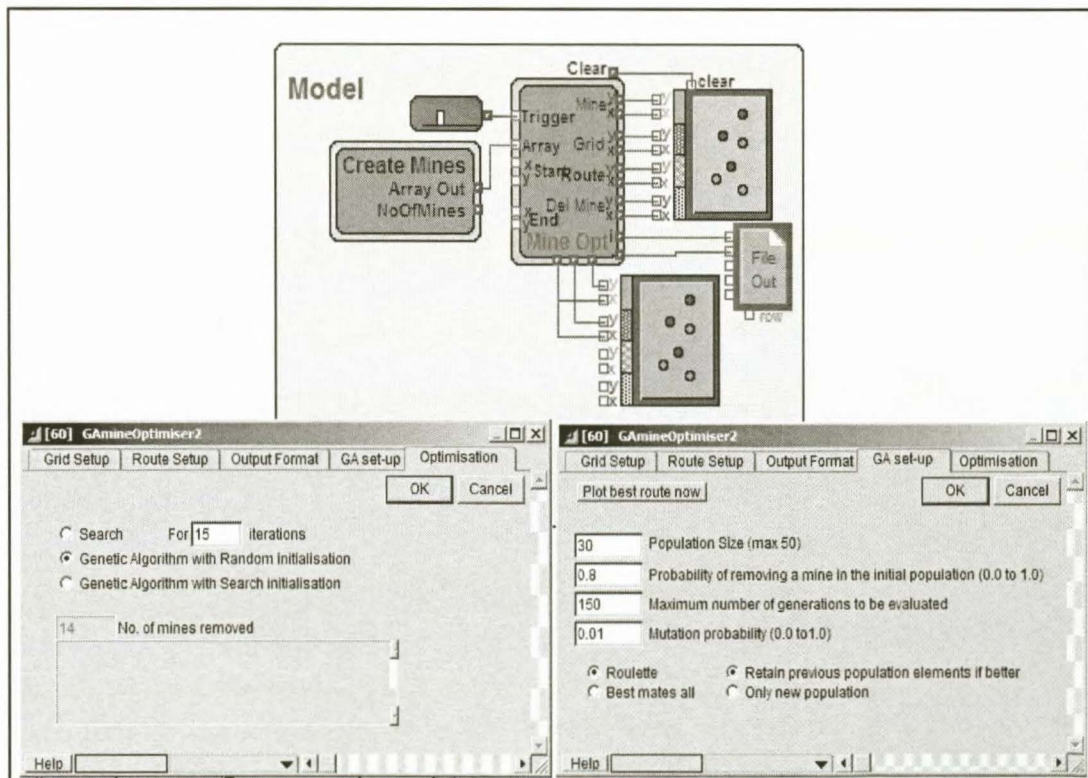


Figure 3-3: Model detail and user interface relating to genetic algorithm parameters



### 3.5 Results of mine removal optimisation implementation

In order to create a denser minefield, which would ensure path blockage for a realistic safety radius, it was decided to use a minefield consisting of 80 mines. To enable a sufficiently dense minefield, the dimensions of the minefield were also reduced. The operational area was reduced to 2000m by 2000m and the minefield 2200m wide and 1600m high, with a buffer of 200m between the top and bottom of the operational area (refer to Figure 2-13 p. 35 for clarity on the layout).

#### 3.5.1 Investigation of crossover and population selection criteria

The parameters described in § 3.3 have been implemented in different combinations and their effect on solution finding and convergence of the GA have been investigated. The results are tabulated in Table 3-1 and give an indication of the GA performance. Because of the time required for the trials only 20 repetitions were done of each of the alternatives. This may be too small a sample from which to draw conclusive statistics, but it is believed that the trends are sufficiently clear.

The population size was set to 20 chromosomes for the results shown here. There were 80 genes (one for each mine) per chromosome in the population, of which 80% were initially randomly set to 0 for each chromosome in order to create the initial population, *i.e.* 64 genes in each chromosome were randomly chosen and set to 0. Because the investigated crossover strategies ensured a variation in selection of partner and survival to the next generation, the crossover probability was selected as 1.0. As the mutation rate is accepted to be in the region of  $1/l$  with  $l$  the length of a chromosome, the mutation probability was set at 0.01.

**Table 3-1: Results of genetic algorithm performance as a function of replication parameters (for 20 repetitions)**

<b>Crossover</b>	<b>Best mates all</b>				<b>Roulette</b>			
<b>Qualification as new parent</b>	<b>Only new generation</b>		<b>Best of previous and new</b>		<b>Only new generation</b>		<b>Best of previous and new</b>	
	<b>Number of mines removed</b>	<b>GA convergence (generations)</b>	<b>Number of mines removed</b>	<b>GA convergence (generations)</b>	<b>Number of mines removed</b>	<b>GA convergence (generations)</b>	<b>Number of mines removed</b>	<b>GA convergence (generations)</b>
<b>Average</b>	11	76.25	10.75	71.8	No convergence	No convergence	9.85	113.2
<b>Worst</b>	18	98	16	96			12	150
<b>Best</b>	8	48	8	52			8	86
<b>No. of best</b>	1		1				1	

Table 3-1 shows that Roulette Wheel selection scores best of all, both in terms of average best result (9.85 *versus* 10.75 and 11) and the worst result obtained (12 *versus* 16 and 18). It does, however, take substantially longer to converge than the less accurate methods (113 generations *versus* 71 and 76). It is noted that Roulette Wheel selection does not converge if only the new generation qualifies as parents for the next generation. It is believed that this can be overcome by selecting the crossover probability to be less than 1, which will enable old generation members to survive to the next generation. It has, however, not been implemented here.

Based on the above results it was decided to implement only that strategy that gave the best results (*Roulette Wheel selection with Best of new and old generation*) for the future evaluation of other parameters. Sample results are shown in the subsequent paragraphs.

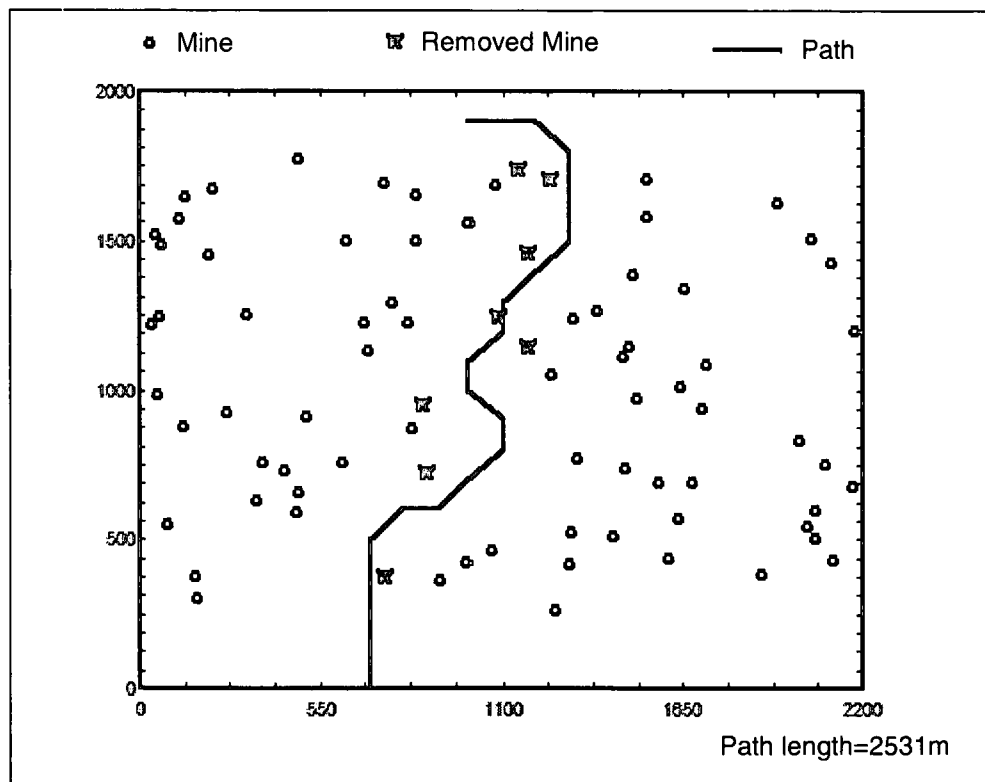
### 3.5.2 Changing the safety distance

Here changing the safety distance is investigated for its effect on the numbers and positions of mines that have to be removed for an acceptable safe path. The method used for the result is a

Roulette Wheel crossover strategy with retention of the parent generation if fitter than the new offspring. The rest of the parameters are as in the previous section.

In Figure 3-4 the result is shown of the best solution found when optimising for the minimum number of mines (8 for this example) to be removed to give a safe passage through the minefield when using a safety radius of 200m. The mines that have to be removed are indicated with crosses. The performance of the GA that determined this solution is shown in Figure 3-5.

The optimisation was repeated for the same random minefield and GA parameters but with a different safety radius. The results in Figure 3-6 and Figure 3-7 show that for a change in safety radius, different mines and a new amount need to be removed, which results in entirely different paths compared to the path suggested in Figure 3-4.



**Figure 3-4: Safe path resulting from 8 mines being removed, following a genetic algorithm optimisation (200m safety radius)**



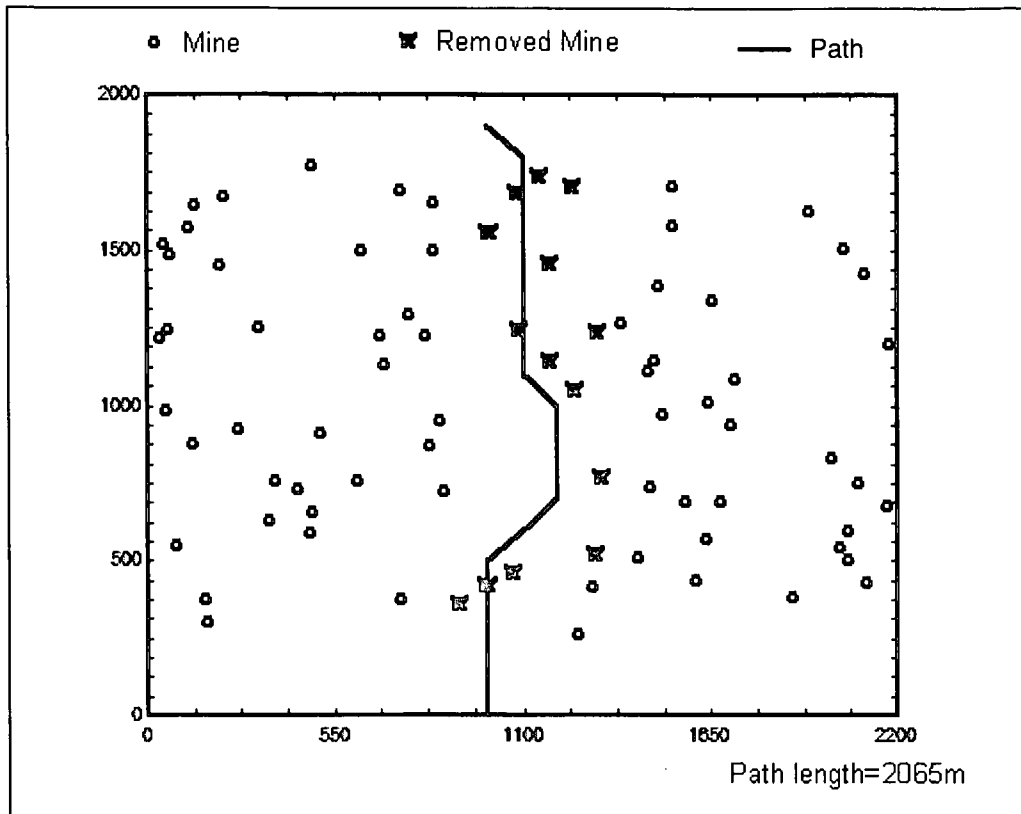


Figure 3-7: New path for a larger safety radius (250m) requiring removal of 14 mines

### 3.5.3 *Results of changing the population size*

The solution time (time to convergence) of the GA is mainly dependent on the number of mines in the minefield (chromosome size) and the population size. As the minefield was fixed for the investigation, here the effect of population size is investigated on the accuracy of the results and the convergence performance. The Roulette Wheel method with selection of new and previous population qualifying as parents for the next generation was implemented with other parameters as before. The results are shown in Table 3-2 where it can be seen that increased population size improves the result accuracy (average of mines to be removed for 20 repeats) but increases the convergence time, although convergence is achieved in fewer generations. From the table it can be suggested that a population of 30 is a good balance between accuracy and solution speed as a larger population has no accuracy advantage but takes longer to converge.



**Table 3-2: Results of GA implemented with different population sizes (for 20 repetitions)**

<b>Population size</b>	<b>10</b>		<b>20</b>		<b>30</b>		<b>40</b>		<b>50</b>	
	<b>Number of mines removed</b>	<b>GA convergence (generations)</b>	<b>Number of mines removed</b>	<b>GA convergence (generations)</b>	<b>Number of mines removed</b>	<b>GA convergence (generations)</b>	<b>Number of mines removed</b>	<b>GA convergence (generations)</b>	<b>Number of mines removed</b>	<b>GA convergence (generations)</b>
<b>Average</b>	10.55	160.8	10.6	112.6	9.5	89.45	9.65	84.7	9.55	68
<b>Worst</b>	14	199	16	140	12	117	12	100	13	77
<b>Best</b>	8	111	8	90	8	74	8	68	8	59
<b>No. of best (out of 20)</b>	1		2		5		4		5	
<b>Time to convergence</b>	48 seconds		67 seconds		72 seconds		93 seconds		95 seconds	

#### 3.5.4 *Results of the random search algorithm*

The random search algorithm as defined in paragraph 3.4 (p. 68) was evaluated and compared to the best performing GA of the previous paragraph and the GA using the initial population generated with random search. The results for 20 repetitions are presented in Table 3-3.

It is evident that the average path generated by the random search is somewhat worse than the standard GA initiated with random populations. The GA initiated with the random search performs extremely well. It is evident, however, from the convergence data that very little value is added by the GA principles. It is more likely that all that is achieved in the GA is that the initial population is sorted according to fitness, and the best of the solutions in the initial population survives to become the final solution. The performance of the GA initiated with the random search is predictably good with 15 out of 20 results giving the minimum of 8 mines to be removed. The other 5 results showed 9 mines to be removed.

**Table 3-3: Results of Random Search Algorithm performance with and without genetic algorithm implementation (for 20 repetitions)**

<i>Optimisation type</i>	<i>Best standard GA</i>		<i>Random Search only</i>		<i>GA initiated by Random Search</i>	
	<i>Number of mines removed</i>	<i>GA convergence (generations)</i>	<i>Number of mines removed</i>	<i>Convergence (generations)</i>	<i>Number of mines removed</i>	<i>GA convergence (generations)</i>
<i>Average</i>	9.85	113.2	12.35	Not Applicable	8.25	5.7
<i>Worst</i>	12	150	22	Not Applicable	9	10
<i>Best</i>	8	86	8	Not Applicable	8	2
<i>No. of best (out of 20)</i>	1		1		15	

### 3.6 Conclusion for mine removal optimisation

In this chapter it was demonstrated that a metaheuristic such as a Genetic Algorithm can be used for the complex combinatorial optimisation problem of selecting the correct combination of mines to be removed so that a minefield can be safely transited. Of the mating strategies used, the Roulette Wheel selection with a “Best mates all” strategy gave the best results. Population size increase ensures quicker convergence, but at a larger solution time penalty. A population size of 30 is suggested for a good balance between the two parameters of speed and accuracy. It was also shown that the implementation of a random search algorithm can substantially improve the speed with which solutions are obtained, when combined with the GA.

## CHAPTER 4 SUMMARY, CONCLUSIONS AND RECOMMENDATIONS

The preceding chapters started by stating a problem that will have to be solved for future mine countermeasures operations to ensure that areas remain open to shipping in times of conflict. A methodology was then proposed and implemented by using combinatorial optimisation methods. In this chapter the loop is closed to the problem formulation by reflecting on the work that was presented, drawing conclusions from results and experiences and subsequently proposing further improvement.

### 4.1 Summary of the research

A methodology has been developed and presented with which the extremely complex and multi-dimensional problem of finding a path of acceptable risk through a charted minefield can be addressed.

This thesis has demonstrated that readily available techniques such as Dijkstra's Algorithm and Genetic Algorithms can be used to solve the combinatorial optimisation problem of finding a sufficiently safe path through a previously charted minefield. The unique combination of both the methods being used in CHAPTER 3 of the thesis is effectively a combinatorial optimisation within another. It is important to note that the shortest path methodology developed in CHAPTER 2 is a prerequisite for the effective solution of the mine neutralisation optimisation that was developed in CHAPTER 3. Also it was shown that the principle of Efficient Frontier Graphs can be usefully applied here to present the relationship between path risk and return.

### 4.2 Conclusions on the research

The most important insight is that risk management in crossing a charted minefield cannot be seen as a simple, single-parameter task. The work performed here has indicated that risk can be managed by a number of different strategies and parameters. It is determined in all instances by the path proximity to mines. However, exact quantification is complex and can be dependent on the following aspects relating to selectable parameters:

- the choice whether risk is made up of a proximity to the closest mine or to a combination of threatening mines, the extreme case which is proximity to all mines;
- the parameter, here labelled  $k$ , that determines the exponent value of the distance between mine and ship, and with which risk is quantified;
- grid resolution, and how this parameter is chosen with regard to the time penalty for selecting a too fine grid;
- introduction of a safety radius which allows optimisation with regards to path length;
- combination of arc risk cost and arc length cost into a single arc risk figure by manipulating the relative contribution of the two parameters.

The following general statements can be made with respect to the work that was performed:

- many of the correct decisions of the above points are dependent on the specifics of the minefield, making it difficult to choose generic values or guidelines that are applicable to all situations;
- it is difficult to generate a single risk figure for paths generated under different risk strategies. This is because the risk figure (and at times even its unit) is a direct result of the risk strategies;
- It is essential to optimise not purely for distance (length) or risk as a much more balanced result can be obtained by including a small percentage of optimisation for the secondary parameter;
- Genetic Algorithms and possibly other meta-heuristic methods can effectively be used to assist with the identification of the best mines to be removed from a minefield to make it possible to safely transit the minefield. The GA can be vastly improved by combining it with a search;

Although the solutions have not been developed here to a level where they can be implemented without further analysis and risk definition, it is evident that the important parameters have been thoroughly considered and presented. The outstanding issues revolve mainly around which of the risk parameters have to be selected. This requires in-depth knowledge of the process of interaction between the sea mine and a ship and is also directly

related to the risk that the decision maker is prepared to take or that the operational constraints place on him.

Conclusions can be made regarding the individual main themes of the thesis. They are stated below.

#### 4.2.1 Path optimisation

For path optimisation it is concluded that risk can be managed in three ways:

- by minimising the total risk figure for a path through the minefield resulting in the *safest path*;
- by stating an acceptable maximum local risk and implementing it through selection of a *safety radius* whereafter the path is minimised for distance, resulting in the *shortest safe path*;
- by combining the two methods above and optimising for a selected balance between *safest* and *shortest* path;

In terms of the effect of the user-selectable parameters required in implementing the routing strategy the following can be concluded:

- for a given minefield, a sensitivity study can ensure that a grid spacing is chosen that matches result accuracy with time-efficient solutions;
- safety radius has little effect on the safest path through a minefield, but the method of gradually increasing the safety radius can be used to establish when minefield closure would result as a function of increasing safety radius. This gives an indication of the closest a ship will pass to a mine when transiting a minefield;

#### 4.2.2 Mine removal

Although acceptable results can be achieved with a standard Genetic Algorithm, it has been shown that a single run will rarely give a best solution. For a large number of repeats the best solution may be found, but at a large time cost. A significant improvement is achieved through initiating the algorithm's starting population through a Random Search Algorithm. The best solution is likely to be found after only a small number of repeats. These repeats require a shorter time to complete than obtaining the equivalent reliability by using the standard GA. This leads to the conclusion that further improvement can be made to the



optimisation by using search methods suited to the specifics of the problem. It is believed that the random search algorithm works reasonably well as the areas sparsely populated by mines are less likely to be repopulated, enabling paths through these sparse areas.

When using the GA, the best results were achieved for a mating strategy that uses Roulette Wheel selection and selects the parents for a next generation from the best candidates out of the old and new population. This method encourages the continued existence of good solutions.

### 4.3 Recommendations for further work

During in-depth study in a field, while solutions are found, additional questions and new problems usually come to the fore and raise the prospect of further improvement. It is believed that sufficient progress has been made in this thesis towards providing answers to the formulated questions. There are additional insights, however, that may be further investigated and if implemented may provide solutions more satisfying than those presented here. Some suggestions that may be considered for further study are given below.

#### 4.3.1 General approach

The general approach or philosophy of solving the least risk path may be improved by minimising the *point of highest risk* as the primary criterion and not the *total path risk* as was done in this thesis.

The point of highest risk in a path through a minefield can be interpreted as the point along the path where the ship passes closest to a mine. To optimise this, it may be preferable to go along a path with a larger cumulative risk value provided it gives a lower *point of highest risk*. The philosophy can thus be changed to one viewing the problem of maximising the distance to the closest mine as a *bottleneck problem* (described in [5] on p. 112). Solving that problem requires optimisation to find the path with the largest bottleneck, *i.e.* with the largest “closest point of approach” to any mine in the minefield. As this point of largest bottleneck can form a part of many different paths, there needs to be a secondary shortest path optimisation, ensuring that of all possible paths passing through the bottleneck, the shortest (or least risk) path is found.

#### 4.3.2 *Dijkstra's Algorithm for path finding*

As has been described, it is essential that the path-finding part of the mine neutralisation optimisation is required to give a quick solution because of the iterative nature of the GA. In [5] (p. 92) an improvement with regard to that aspect is given in the form of Dijkstra's Two-Tree Algorithm where paths are generated simultaneously from the start- and end-node. The indication of performance improvement is not given in time units, but for a stated example improves from requiring 50% of nodes to be scanned for conventional Dijkstra to 6% for the Two-Tree Algorithm. As solution time is directly related to the number of nodes that have to be scanned, this may relate to a large reduction of solution time. Although not specifically described, it is imaginable that in the A\* algorithm, using a heuristic which relates the shortest distance between currently scanned nodes, can further improve the standard Dijkstra's Two-Tree Algorithm. These improvements could greatly assist the speed and accuracy of the GA implementation described in CHAPTER 3. It may also make it more feasible to use the *safest path* approach of evaluating solution fitness in the Genetic Algorithm used to optimise mine removal instead of the *shortest path* that was used.

#### 4.3.3 *Mine removal optimisation*

It was indicated how well a random search works at identifying feasible mines for removal. It seems logical that a more intelligent search could be more successfully used either to find the best mines to remove or to create more varied initial populations for a GA. The search to identify mines could well be based on identifying areas in the minefield that are sparsely populated and to identify mines that form barriers to these areas. The logic behind this is that a safe passage is most likely in sparsely populated areas, but these areas need to be connected to each other and to the start and end nodes.

An alternative way of generating paths through the minefield may be by generating paths into the minefield as far as possible to and from the start- and end-nodes. These sections of paths then have to be examined to determine which mines obstruct them from meeting to form an unimpeded path. The two path sections that require the least mines removed for them to meet, would be the preferred path. This search method is entirely unrelated to those discussed here and would require a substantial change to Dijkstra's Algorithm to enable it to identify incomplete paths. The method could also be used for the generation of a starting population for a modified GA.

## REFERENCES

- [1] Chasomeris, M G: *South Africa's Port Performance: Policy, Pricing and Growth*, Department of Economics and Finance, University of Kwa-Zulu Natal, 2005.
- [2] Levie, H S: *Mine Warfare at Sea*, Martinus Nijhoff Publishers, The Netherlands, 1992.
- [3] Hartmann, G K; Truver, S C: *Weapons that Wait, Mine Warfare in the US Navy*, Updated Edition, United States Naval Institute, Maryland, 1991.
- [4] Russel S; Norvig P; ed.: *Artificial Intelligence: A Modern Approach*, Prentice Hall, 1995.
- [5] Evans, J R; Minieka E: *Optimisation Algorithms for Networks and Graphs*, 2<sup>nd</sup> ed. Revised, Marcel Dekker Inc, New York, 1991, pp 77-122.
- [6] Marti, J; Bunn, C: *Automated Path Planning for Simulation*, [Online], [Cited on 20<sup>th</sup> June 2002] available from  
<http://www.computer.org/conferences/ais94/marty/paper.html>
- [7] Antonelli G; Chiaverini S; Finotello R; Schiavon R: *Real-Time Path Planning and Obstacle Avoidance for RAIS: An Autonomous Underwater Vehicle*, IEEE Journal of Oceanic Engineering, Vol 26, No 2, April 2001.
- [8] Mitchell, JSB: *An Algorithmic Approach to Some Problems in Terrain Navigation*, Artificial Intelligence, Elsevier Science Publishers, 1988, pp 171-201.
- [9] Boerman, D A: *Finding an Optimal Path Through a Mapped Minefield*, MSc Thesis, Naval Postgraduate School, Monterey, California, March 1994.
- [10] *Graph Theory*, [Online], [Cited on 12<sup>th</sup> December 2005] available from  
[http://en.wikipedia.org/wiki/Graph\\_theory](http://en.wikipedia.org/wiki/Graph_theory)
- [11] Patel AJ: *Amit's Thoughts on Path-Finding and A-Star*, [Online], [Cited on 23<sup>rd</sup> January 2006] available from <http://theory.stanford.edu/~amitp/GameProgramming/>
- [12] Engineering Applications of OR, *The shortest Path Problem*, [Online], [Cited on 23<sup>rd</sup> September 2002] available from  
<http://www.orie.cornell.edu/~or115/handouts/handout3/handout3.html>
- [13] Morris J: *Data Structures and Algorithms: Operation of Dijkstra's Algorithm*, [Online], [Cited on 23<sup>rd</sup> January 2006] available from  
<http://oopweb.com/Algorithms/Documents/PLDS210/Volume/dij-op.html>

- [14] Portfolio Theory, [Online], [Cited on 28<sup>th</sup> November 2005] available from [http://www.riskglossary.com/link/portfolio\\_theory.htm](http://www.riskglossary.com/link/portfolio_theory.htm)
- [15] Efficient Frontier, [Online], [Cited on 28<sup>th</sup> November 2005] available from [http://www.riskglossary.com/link/efficient\\_frontier.htm](http://www.riskglossary.com/link/efficient_frontier.htm)
- [16] Modern Portfolio Theory and the Efficient Frontier, [Online], [Cited on 28<sup>th</sup> November 2005] available from <http://www.gummy-stuff.org/Efficient-Frontier.htm>
- [17] Efficient Frontier, [Online], [Cited on 28<sup>th</sup> November 2005] available from <http://www.investopedia.com/terms/e/efficientfrontier.asp>
- [18] Eklund PW; Kirby S; Pollitt S: *A Dynamic Multi-Source Dijkstra's Algorithm for Vehicle Routing*, **Published:** *Australian and New Zealand Conference on Intelligent Information Systems (ANZIIS '96)*, pp.329-333, IEEE Press, 1996
- [19] ENGINEER'S HANDBOOK - *US Navy Salvage Engineers Handbook, Volume 1- Salvage Engineering, Chapter 10: Explosions*, S0300-A8-HBK-010, 1992. [Online], [Cited on 28<sup>th</sup> November 2005] available from <http://www.kvocentral.org/kvopapers/pollitt.pdf>
- [20] Gogg TJ; Mott JRA: *Improve Quality & Productivity with Simulation*, 3<sup>rd</sup> ed. JMIconulting Group, 1996.
- [21] Hoff A; Løketangen A; Mittet I: *Genetic Algorithms for 0/1 Multidimensional Knapsack Problems*, Proceedings Norsk Informatikk Konferanse, 1996.
- [22] Phutan Veettil SK: *A Genetic Algorithm for 0/1 Knapsack Problem*, CS 704: Analysis of Algorithms.
- [23] Khuri S; Bäck T; Heitkötter J: *The Zero/One Multiple Knapsack Problem and Genetic Algorithms*, ACM Symposium of Applied Computation (SAC'94) proceedings, 1993 ACM Press.
- [24] Goldberg DE: *Genetic Algorithms in Search, Optimisation, and Machine Learning*, Addison-Wesley, 1989.

APPENDIX A: COMPUTER LISTING OF *ROUTEOPTIMISER* BLOCK

**NOTE:** While care has been taken to ensure code correctness, there was no formal verification done of the code other than reported in this thesis. As such the functional correctness can not be assumed by someone copying this code and such a person will have to take full responsibility for ensuring the code given here fulfils their functional requirements.

```

real      nodeXY[][2]; //array for x any co-ordinates of each node as well as max and summed risk of each
node
real      link[][9]; //array for valid links from each node
real      risk[][10]; //array for associated risk of each valid link
real      length[][10]; //array for associated length of each valid link
real      dummyArray[][8]; //used to receive mine positions
real      shortest[][5]; //used to store the distance from each node to the end node and preceding node
integer nodes, first, mines, noRoute, connected;

procedure findRoute()
{
  integer i,j,k,l,riskSelect, currentNode, thisLink, nextNode, lastNode;
  integer StartNode, EndNode, count, iterations;
  real startDist, bestStartDist, endDist, bestEndDist, closestToStart, closestToEnd;
  real xCoord, yCoord, nodeRisk, sumRisk, maxRisk;
  real arclength, arcRisk, riskValue, thisRisk, thisLength, lowestArc, dSquared, dist;
  real xStart, yStart, xEnd, yEnd, xIncr, yIncr, segmentRisk;
  real bestRisk, bestLength;

  Result="";
  totalLength=blank;
  StartNode=0;
  EndNode=0;
  bestStartDist=100000000;
  bestEndDist=100000000;
  **check start and end nodes correct
  count=0;
  comments="Start";
  first=true;
  nodes=rows*cols;
  bestRisk=0;
  bestLength=0;
  ** get values if connected
  if(connected)
  {
    sendMsgToAllCons(xStartIn);
    sendMsgToAllCons(yStartIn);
    sendMsgToAllCons(xEndIn);
    sendMsgToAllCons(yEndIn);
  }
  **get array with mine co-ordinates
  if (getPassedArray(minesIn, dummyArray))
  {
    ** The array was passed successfully
  }
  else ** FALSE result means the array was not passed yet.

```



```

    {
        usererror("No Mine array received");
        abort;
    }
mines=getDimension(dummyArray);
//plot mine positions
routeXout=blank;
routeYout=blank;
for i=0 to mines-1
    {
        mineXout=dummyArray[i][0];
        mineYout=dummyArray[i][1];
        sendMsgToInputs(MineXout);
        if(damageR) //plot damage radius if ruquired
            {
                for (j=0; j<360; j=j+3)
                    {
                        xOut=mineXout+safeDist*sin(j*Pi/180);
                        yOut=mineYout+safeDist*cos(j*Pi/180);
                        sendMsgToInputs(xOut);
                    }
                Xout=blank;
                Yout=blank;
            }
        if (allXY)
            sendMsgToInputs(MineYout);
    }
** create and populate the array with node co-ordinates: col0=x co-ord, col1=y co-ord,
makearray(nodeXY, nodes+2);
for i=0 to 1
    {
        nodeXY[0][i]=Blank;
        nodeXY[nodes+1][i]=Blank;
    }
for i=1 to nodes
    {
        xCoord=(((i-1) mod cols)*spacing)+xOff;
        yCoord=((int((i-1)/cols))*spacing)+yOff;
        nodeXY[i][0]=xCoord;
        nodeXY[i][1]=yCoord;
        **find nodes closest to start and end co-ordinates
        if(connected)
            {
                startDist=((xStartIn-xCoord)^2+(yStartIn-yCoord)^2)^0.5;
                if(startDist<bestStartDist)
                    {
                        bestStartDist=startDist;
                        entryFirst=i;
                    }
                endDist=((xEndIn-xCoord)^2+(yEndIn-yCoord)^2)^0.5;
                if(endDist<bestEndDist)
                    {
                        bestEndDist=endDist;
                        exitFirst=i;
                    }
            }
    }
}
** create and poulate the array with all possible node links
makearray(link, nodes+2);
for i=1 to nodes

```

```

{
**all possible adjacent nodes
link[i][0]=i-cols-1;
link[i][1]=i-cols;
link[i][2]=i-cols+1;
link[i][3]=i-1;
link[i][4]=i+1;
link[i][5]=i+cols-1;
link[i][6]=i+cols;
link[i][7]=i+cols+1;
**erase illegal nodes for first column
if((i-1)mod cols==0)
{
link[i][0]=blank;
link[i][3]=blank;
link[i][5]=blank;
}
**erase illegal nodes for bottom row
if((i-cols)<=0)
{
link[i][0]=blank;
link[i][1]=blank;
link[i][2]=blank;
}
**erase illegal nodes for last column
if((i)mod cols==0)
{
link[i][2]=blank;
link[i][4]=blank;
link[i][7]=blank;
}
**erase illegal nodes for top row
if((i+cols)>nodes)
{
link[i][5]=blank;
link[i][6]=blank;
link[i][7]=blank;
}
**Add Start and Exit nodes links if they are a range. Nodes+1 is the start node, 0 is the exit node
if (entryLast and i>=entryfirst and i<=entryLast) //if i is in start node range
link[i][8]=nodes+1;
else
link[i][8]=blank;
}
for i=0 to 8
{
link[0][i]=blank;
if(exitLast)
{
if(exitFirst+i<=exitLast)
link[0][i]=exitFirst+i; //set range of exit links to node 0
}
link[nodes+1][i]=blank; //no link from nodes+1 to any other node
}

**Create and populate an array containing all the risks for valid links
comments="Initiating: creating risk matrix";
makearray(risk,nodes+2);
makearray(length,nodes+2);

```

```

if (accuracy==1)
    iterations=11;
else
    iterations=1;
for i=0 to nodes+1
    {
    if (algorithm==1) //distance from all nodes to start point for A* heuristic
        risk[i][9]=((nodeXY[i][0]-nodeXY[entryFirst][0])^2+
                    (nodeXY[i][1]-nodeXY[entryFirst][1])^2)^0.5;
    for j=0 to 8
        {
        if(link[i][j]) //link exists
            {
            if(i==0 or j==8) //risks and lengths of links from nodes 0 and to nodes+1 =0
                {
                risk[i][j]=0;
                length[i][j]=0;
                }
            else
                {
                arcLength=((nodeXY[i][0]-nodeXY[link[i][j]][0])^2+
                            (nodeXY[i][1]-nodeXY[link[i][j]][1])^2)^0.5;
                xStart=nodeXY[i][0];
                yStart=nodeXY[i][1];
                xEnd=nodeXY[link[i][j]][0];
                yEnd=nodeXY[link[i][j]][1];
                xIncr=(xEnd-xStart)/10;
                yIncr=(yEnd-yStart)/10;
                if (accuracy==2)
                    {
                    xStart=xStart+5*xIncr;
                    yStart=yStart+5*yIncr;
                    }
                segmentRisk=0;
                maxRisk=0;
                for k=1 to iterations
                    {
                    for l=0 to mines-1
                        {
                        dSquared=((dummyArray[l][0]-xStart)^2+(dummyArray[l][1]-yStart)^2);
                        dist=dSquared^0.5;
                        if(dist<SafeDist) //if distance is less than safety distance
                            {
                            link[i][j]=blank; //erase unsafe link
                            risk[i][j]=blank;
                            goto nextLink; //exit to next link
                            }
                        switch(mineRisk) //calculate arc risk
                            {
                            case 1: //closest mine
                                segmentRisk=dist^riskExponent;
                                if(segmentRisk>maxRisk)
                                    maxRisk=segmentRisk;
                                segmentRisk=maxRisk;
                                break;
                            case 2: //all mines
                                segmentRisk=segmentRisk+dist^riskExponent;
                                break;
                            }
                        }
                    }
                }
            }
        }
    }
}

```

```

        xStart=xStart+xIncr;
        yStart=yStart+yIncr;
    }
    risk[i][j]=segmentRisk*arclength;
    length[i][j]=arcLength;
    if(risk[i][j]>bestRisk)
        bestRisk=Risk[i][j];
    if(length[i][j]>bestLength)
        bestLength=length[i][j];
    }
}
else //link does not exist
{
    risk[i][j]=blank;
    length[i][j]=blank;
}
nextLink:
}
}
risk[nodes+1][9]=0; //for multiple start nodes risk is 0 to start node
length[nodes+1][9]=0; //for multiple start nodes distance is 0 to start node
comments=count;

** Implementation of Dijkstra's/A* algorithm
**Create Array for distances to end node from each node
makeArray(shortest,nodes+2);
for i=0 to nodes+1
{
    count++;
    shortest[i][0]=100000000000; //initialise all to infinite
    shortest[i][1]=blank;
    shortest[i][2]=blank;
    shortest[i][3]=blank;
    shortest[i][4]=0;
}
if(exitLast)
    nextNode=0;
else
    nextNode=exitFirst;
if (entryLast)
    lastNode=nodes+1;
else
    lastNode=entryFirst;
shortest[nextNode][0]=0; //set end node to 0
shortest[nextNode][1]=0; //final length to 0
shortest[nextNode][2]=0; //predecessor to 0
shortest[nextNode][3]=0; //risk to 0
currentNode=-1;
thisLink=0;
**find shortest link from current node to all linked nodes
while (currentNode != lastNode) //until start node has been scanned
{
    currentNode=nextNode;
    lowestArc=1000000000000;
    for i=0 to 8 //loop to check for all possible links to currentNode and update paths
    {
        count++;
        thisLink=link[currentNode][i];
        if (!novalue(link[currentNode][i]) and shortest[thisLink][0]) //check if a link exists and the
endNode has not been removed

```

```

    {
//from shortest Array col0 because it is on the shortest route
    thisLength=length[currentNode][i]/bestLength+shortest[currentNode][4];
    riskValue=risk[currentNode][i]/bestRisk+shortest[currentNode][3];
    thisRisk=corrFact*risk[currentNode][i]/bestRisk+shortest[currentNode][0];

    thisRisk=thisLength*riskPercent/100+thisRisk*(1-riskPercent/100);
    if (thisRisk<shortest[thisLink][0]) //check if this path is shorter
    {
        shortest[thisLink][0]=thisrisk; //update value for optimising
        shortest[thisLink][3]=riskValue; //update risk to endNode
        shortest[thisLink][4]=thisLength; //update distance to endNode
        shortest[thisLink][2]=currentNode; //update predecessor endNode
    }
    }
}
shortest[currentNode][1]=shortest[currentNode][0];
shortest[currentNode][0]=blank;
if (algorithm==1) //for A*
    {
        for k=0 to nodes+1 //find shortest arc and make it the next node
        {
            if (!noValue(shortest[k][0]) and shortest[k][0]+risk[k][9]<lowestArc)
            {
                lowestArc=shortest[k][0]+risk[k][9];
                nextNode=k;
            }
            count++;
        }
    }
else //for Dijkstra
    {
        for k=0 to nodes+1 //find shortest arc and make it the next node
        {
            if (!noValue(shortest[k][0]) and shortest[k][0]<lowestArc)
            {
                lowestArc=shortest[k][0];
                nextNode=k;
            }
            count++;
        }
    }
}
comments=count;

**Trace back the shortest route from the start node
noRoute=false;
currentNode=-1;
// Sort out start and end nodes as a function of multiple nodes selected
if (entryLast)
    nextNode=nodes+1;
else
    nextNode=entryFirst;
if (exitLast)
    lastNode=0;
else
    lastNode=exitFirst;
i=0;
safetyOut=riskPercent;
riskOut=shortest[nextnode][3]*bestRisk;

```



```

distOut=shortest[nextnode][4]*bestLength;
SendMsgToInputs(safetyOut);
Result=riskOut;
TotalLength=distOut;
while (currentNode<>lastNode)
    {
    i++;
    if(i>nodes+1)
        {
        noRoute=true;
        Result="No Route";
        return;
        }
    currentNode=nextNode;
    nextNode=shortest[currentNode][2];
    shortest[i][2]=currentNode;
    }
shortest[i+1][2]=blank;
}

```

\*\* This messagehandler ensures the route is only calculated when a trigger is received on triggerIn

```

{
integer i;
if(triggerIn)
{
if(first)
{
sendMsgToOutputs(safetyIn);
if(safetyIn)
    riskPercent=safetyIn;

    findRoute(); //do procedure to find the route
if(frontier)
    return;
first=false;
xOut=blank;
yOut=blank;
routeXout=blank;
routeYout=blank;
mineXout=blank;
mineYout=blank;
for i=1 to nodes
    {
    if(!shortest[i][2]) //break if a noValue is found
        break;
    pointOut=i;
    RouteXout=nodeXY[shortest[i][2]][0];
    RouteYout=nodeXY[shortest[i][2]][1];
    sendMsgToInputs(RouteXOut);
    if (allXY)
        {
        sendMsgToInputs(routeYOut);
        }
    }
}
if (gridOn)
    {
    for i=1 to nodes
        {
        xOut=nodeXY[i][0];

```

```

        yOut=nodeXY[i][1];
        sendMsgToInputs(xOut);
    }
}
RouteXout=blank;
RouteYout=blank;
xOut=blank;
yOut=blank;
}
}
}

```

\*\*for auto-updating the start and end node to centre of top and bottom row  
on update

```

{
entryFirst=(cols/2);
exitFirst=(rows*cols)-(cols/2);
entryLast=blank;
exitLast=blank;
}

```

\*\*check input data for completeness and correctness  
on checkdata

```

{
integer dataError;
dataError=false;
connected=false;
nodes=rows*cols;
if(xStartIn or yStartIn or xEndIn or yEndIn)
{
connected=true;
if(!xStartIn)
{
usererror("x Start co-ordinate is not connected");
dataError=true;
}
if(!yStartIn)
{
usererror("y Start co-ordinate is not connected");
dataError=true;
}
if(!xEndIn)
{
usererror("x End co-ordinate is not connected");
dataError=true;
}
if(!yEndIn)
{
usererror("y End co-ordinate is not connected");
dataError=true;
}
if(dataError)
{
usererror("Connect input connectors correctly!!");
abort;
}
}
}
if (entryLast) //A* won't work with multiple starting nodes

```

```
{
  if(algorithm==1)
    usererror("A* algorithm does not work with multiple starting nodes.Using Dijkstra instead!");

  Algorithm=2;
}
if(!entryFirst or !exitFirst)
{
  usererror("Start/end node not specified!!");
  abort;
}
if (EntryFirst>nodes or exitFirst>nodes or entryLast>nodes or exitLast>nodes)
{
  usererror("Start/end node greater than number of nodes!! Max allowed is "+nodes);
  abort;
}
if (exitLast and exitLast-exitFirst>8 )
{
  usererror("No more than 8 nodes can be specified as Exit Nodes!!");
  abort;
}
}

** Initialize any simulation variables.
on initsim
{
  first=true;
  routeXout=blank;
  routeYout=blank;
}

**Message handlers to prevent excessive message emulation
on xOut
{
}

on yOut
{
}

on routexOut
{
}

on routeyOut
{
}

on minexOut
{
}

on mineyOut
{
}
```

APPENDIX B: COMPUTER LISTING OF *GAMINEOPTIMISER* BLOCK

**NOTE:** While care has been taken to ensure code correctness, there was no formal verification done of the code other than reported in this thesis. As such the functional correctness can not be assumed by someone copying this code and such a person will have to take full responsibility for ensuring the code given here fulfils their functional requirements.

```
//variables for route finding part
real  nodeXY[][2]; //array for x and y co-ordinates of each node as well as max and summed risk of each node
real  nodeDistance[][100]; //array that records the nodes distance to each mine (100 mines maximum)
real  nodeErase[][100]; //array that records the nodes that are erased by each mine because of safety
distance(100 mines maximum)
real  link[][9]; //array for valid links from each node
real  length[][10]; //array for associated length of each valid link
real  OrigArray[][8]; //used to receive original mine positions
real  dummyArray[][3]; //used to create updated mine positions
real  shortest[][5]; //used to store the distance from each node to the end node and preceding node
integer  initTicks,NoOfTicks; //used for checking timings
integer  nodes, first, origMines, mines, noRoute, connected;
//variables for Genetic Algorithm part
real  CurrentPop[][101]; //array showing mines present (1) and mines removed (0) in current generation, fitness
in last column
real  bestPath;
integer  plot;
integer  minesAvail[][1]; //used for search initialisation and optimisation

on plotNow
{
  plot=TRUE;
}

on update
{
  entryFirst=(cols/2);
  exitFirst=(rows*cols)-(cols/2);
  entryLast=blank;
  exitLast=blank;
}

on stepsize
{
  comments="Init";
  InitTicks=tickCount();
}

on finalCalc
{
  comments=comments+(tickCount()-InitTicks);
  pointOut=delMines;
  sendMsgToInputs(pointOut);
}
```

```

on checkdata
{
integer dataError;
dataError=false;
connected=false;
nodes=rows*cols;

if(xStartIn or yStartIn or xEndIn or yEndIn)
{
connected=true;
if(!xStartIn)
{
usererror("x Start co-ordinate is not connected");
dataError=true;
}
if(!yStartIn)
{
usererror("y Start co-ordinate is not connected");
dataError=true;
}
if(!xEndIn)
{
usererror("x End co-ordinate is not connected");
dataError=true;
}
if(!yEndIn)
{
usererror("y End co-ordinate is not connected");
dataError=true;
}
if(dataError)
{
usererror("Connect input connectors correctly!!");
abort;
}
}
if(!entryFirst or !exitFirst)
{
usererror("Start/end node not specified!!");
abort;
}
if (EntryFirst>nodes or exitFirst>nodes or entryLast>nodes or exitLast>nodes)
{
usererror("Start/end node greater than number of nodes!! Max allowed is "+nodes);
abort;
}
if (exitLast and exitLast-exitFirst>8 )
{
usererror("No more than 8 nodes can be specified as Exit Nodes!!");
abort;
}
}

** Initialize any simulation variables.
on initsim
{
delMineXout=blank;
delMineYout=blank;
first=true;

```

```

plot=FALSE;
}

procedure findRoute()
{
  integer i,j,k,l,riskSelect, currentNode, thisLink, nextNode, lastNode;
  integer StartNode, EndNode, count, iterations;
  real startDist, bestStartDist, endDist,bestEndDist, closestToStart, closestToEnd,
  xCoord,yCoord,sumRisk,maxRisk;
  real arclength, arcRisk,thisRisk,thisLength,lowestArc,dSquared;
  real xStart, yStart, xEnd, yEnd, xIncr, yIncr, segmentRisk;

  Result=100000000;
  totalLength=blank;
  StartNode=0;
  EndNode=0;
  bestStartDist=100000000;
  bestEndDist=100000000;
  **check start and end nodes correct
  count=0;

  noOfTicks=TickCount()-InitTicks;
  comments=(noOfTicks)+" Nodes ";

  mines=getDimension(dummyArray);

  **do only once
  if(First)
  {
    first=False;
    ** create and populate the array with node co-ordinates: col0=x co-ord, col1=y co-ord,
    makearray(nodeXY, nodes+2);

    for i=0 to 1
    {
      nodeXY[0][i]=Blank;
      nodeXY[nodes+1][i]=Blank;
    }
    for i=1 to nodes
    {
      // count++;
      xCoord=(((i-1) mod cols)*spacing)+xOff;
      yCoord=((int((i-1)/cols))*spacing)+yOff;
      nodeXY[i][0]=xCoord;
      nodeXY[i][1]=yCoord;
      **find nodes closest to start and end co-ordinates
      if(connected)
      {
        startDist=((xStartIn-xCoord)^2+(yStartIn-yCoord)^2)^0.5;
        if(startDist<bestStartDist)
        {
          bestStartDist=startDist;
          entryFirst=i;
        }
        endDist=((xEndIn-xCoord)^2+(yEndIn-yCoord)^2)^0.5;
        if(endDist<bestEndDist)
        {
          bestEndDist=endDist;
          exitFirst=i;
        }
      }
    }
  }
}

```



```

    }
  }
  **create array with mine risk of each node , and array of distance between each mine and node
  disposeArray(nodeDistance);
  disposeArray(nodeErase);
  makeArray(nodeDistance,nodes+1);
  makeArray(nodeErase,nodes+1);
  for i=1 to nodes
  {
    for j=0 to mines-1
    {
      dSquared=((nodeXY[i][0]-dummyArray[j][0])^2+(nodeXY[i][1]-dummyArray[j][1])^2);
      arcLength=dSquared^0.5;
      nodeDistance[i][j]=arcLength;
      if (arcLength<safeDist) //mark node for erasing if it is closer than safe distance to a mine
        nodeErase[i][j]=1;
    }
  }
}

** create and populate the array with all possible node links
makearray(link, nodes+2);
makearray(length,nodes+2);
noOfTicks=TickCount()-InitTicks;
comments=comments+(noOfTicks)+", Links ";
for i=1 to nodes
{
  **all possible adjacent nodes
  link[i][0]=i-cols-1;
  link[i][1]=i-cols;
  link[i][2]=i-cols+1;
  link[i][3]=i-1;
  link[i][4]=i+1;
  link[i][5]=i+cols-1;
  link[i][6]=i+cols;
  link[i][7]=i+cols+1;

  length[i][0]=2^0.5*spacing;
  length[i][1]=spacing;
  length[i][2]=2^0.5*spacing;
  length[i][3]=spacing;
  length[i][4]=spacing;
  length[i][5]=2^0.5*spacing;
  length[i][6]=spacing;
  length[i][7]=2^0.5*spacing;
  **erase illegal nodes for first column
  if((i-1)mod cols==0)
  {
    link[i][0]=blank;
    link[i][3]=blank;
    link[i][5]=blank;
  }
  **erase illegal nodes for bottom row
  if((i-cols)<=0)
  {
    link[i][0]=blank;
    link[i][1]=blank;
    link[i][2]=blank;
  }
  **erase illegal nodes for last column

```

```

if((i)mod cols==0)
{
link[i][2]=blank;
link[i][4]=blank;
link[i][7]=blank;
}
**erase illegal nodes for top row
if((i+cols)>nodes)
{
link[i][5]=blank;
link[i][6]=blank;
link[i][7]=blank;
}
**Add Start and Exit nodes links if they are a range. Nodes+1 is the start node, 0 is the exit node
if (entryLast and i>=entryfirst and i<=entryLast) //if i is in start node range
{
link[i][8]=nodes+1;
length[i][8]=0;
}
else
link[i][8]=blank;
}
for i=0 to 8
{
link[0][i]=blank;
length[0][i]=0;
if(exitLast)
{
if(exitFirst+i<=exitLast)
link[0][i]=exitFirst+i; //set range of exit links to node 0
}
link[nodes+1][i]=blank; //no link from nodes+1 to any other node
}

**delete links with illegal nodes
noOfTicks=TickCount()-InitTicks;
comments=comments+(noOfTicks)+", Risk ";
for l=0 to mines-1 //for each mine
{
if(dummyArray[l][2]==1) //if mine is there check all nodes that need to be erased
{
for i=0 to nodes //for each node
{
if(nodeErase[i][1]==1) //if node is marked for erasing
{
for j=0 to 8 //for each link of the node to be erased
{
if(!noValue(link[i][j]))
{
for k=0 to 8 //erase reciprocal link
{
if(link[link[i][j]][k]==i)
link[link[i][j]][k]=blank;
}
link[i][j]=blank; //erase unsafe link
}
}
}
}
}
}
}

```

```

    }
  }
  length[nodes+1][9]=0; //for multiple start nodes distance is 0 to start node

** Implementation of Dijkstra's algorithm
noOfTicks=TickCount()-InitTicks;
comments=comments+(noOfTicks)+", Alg ";

makeArray(shortest,nodes+2);
for i=0 to nodes+1
  {
  count++;
  shortest[i][0]=10000000000; //initialise all to infinite
  shortest[i][1]=blank;
  shortest[i][2]=blank;
  shortest[i][3]=blank;
  shortest[i][4]=0;
  }
if(exitLast)
  nextNode=0;
else
  nextNode=exitFirst;
if (entryLast)
  lastNode=nodes+1;
else
  lastNode=entryFirst;
shortest[nextNode][0]=0; //set end node to 0
shortest[nextNode][1]=0; //final length to 0
shortest[nextNode][2]=0; //predecessor to 0
currentNode=-1;
thisLink=0;
**find shortest link from current node to all linked nodes
while (currentNode != lastNode) //until start node has been scanned
  {
  currentNode=nextNode;
  lowestArc=10000000000;
  for i=0 to 8 //loop to check for all possible links to currentNode and update paths
    {
    count++;
    thisLink=link[currentNode][i];
    if (!noValue(thisLink) and shortest[thisLink][0]) //check if a link exists and the endNode has not been
removed
      {
      thisLength=length[currentNode][i]+shortest[currentNode][4];
      if (thisLength<shortest[thisLink][0]) //check if this path is shorter
        {
        shortest[thisLink][0]=thisLength; //update risk to endNode
        shortest[thisLink][4]=thisLength; //update distance to endNode
        shortest[thisLink][2]=currentNode; //update predecessor of this endNode
        }
      }
    }
  }
  shortest[currentNode][1]=shortest[currentNode][0];
  shortest[currentNode][0]=blank;
  //find shortest arc and make it the next node
  nextNode=-1;
  for k=0 to nodes+1
    {
    if (!noValue(shortest[k][0]) and shortest[k][0]<lowestArc)
      {

```

```

        lowestArc=shortest[k][0];
        nextNode=k;
    }
    count++;
}
if(nextNode==-1) //no route was found because there is no suitable nextNode
{
    noRoute=True;
    return;
}
}
noOfTicks=TickCount()-InitTicks;
comments=comments+(noOfTicks)+" , Route ";

**Trace back the shortest route from the start node
noRoute=false;
currentNode=-1;
// Sort out start and end nodes as a function of multiple nodes selected
if (entryLast)
    nextNode=nodes+1;
else
    nextNode=entryFirst;
if (exitLast)
    lastNode=0;
else
    lastNode=exitFirst;
i=0;
Result=shortest[nextnode][1];
TotalLength=shortest[nextnode][4];
while (currentNode<>lastNode)
{
    i++;
    currentNode=nextNode;
    nextNode=shortest[currentNode][2];
    shortest[i][3]=currentNode;
}

noOfTicks=TickCount()-InitTicks;
comments=comments+(noOfTicks)+" , Finish ";
}
procedure plotBest()
{
    integer i,j;
    //send message to clear plot
    plotOut=true;
    sendMsgToInputs(plotOut);
    plotOut=False;
    sendMsgToInputs(plotOut);
    **do last Route find with the best solution (top row in array) in order to plot it out
    if (gridOn)
    {
        for i=1 to nodes
        {
            xOut=nodeXY[i][0];
            yOut=nodeXY[i][1];
            sendMsgToInputs(xOut);
        }
    }
    for j=0 to mines-1
    {

```

```

    if(currentPop[0][j]==1)
        dummyArray[j][2]=1;
    else
        {
            dummyArray[j][2]=0;
        }
    }
}
findRoute();
//plot mine positions
routeXout=blank;
routeYout=blank;
for i=0 to origMines-1
    {
        mineXout=origArray[i][0];
        mineYout=origArray[i][1];
        sendMsgToInputs(MineXout);
        if (allXY)
            sendMsgToInputs(MineYout);
    }
mineXout=blank;
mineYout=blank;
//plot deleted mines
for j=0 to origMines-1
    {
        if (currentPop[0][j]==0)
            {
                DelMineXout=origArray[j][0];
                DelMineYout=origArray[j][1];
                sendMsgToInputs(DelMineXout);
                if(allXY)
                    sendMsgToInputs(DelMineYout);
            }
    }
sendMsgToInputs(DelMineXout); //to ensure last deleted mine is displayed
delMineXout=blank;
delMineYout=blank;
**plot the route
RouteXout=blank;
RouteYout=blank;
sendMsgToInputs(RouteXOut);
for i=1 to nodes
    {
        if(!shortest[i][3]) //break if a noValue is found
            break;
        pointOut=i;
        RouteXout=nodeXY[shortest[i][3]][0];
        RouteYout=nodeXY[shortest[i][3]][1];
        sendMsgToInputs(RouteXOut);
        if (allXY)
            {
                sendMsgToInputs(routeYOut);
            }
    }
RouteXout=blank;
RouteYout=blank;
sendMsgToInputs(RouteXOut);
}

procedure search() //used for initial population for GA and search optimisation
{

```

```

integer i,j,k,minesLeft,cut,replace,next,delMinesTemp;
real fitness;

makeArray(minesAvail,mines);
for j=0 to mines-1 //check each mine
  {
    minesAvail[j][0]=j;
    currentPop[0][j]=0; //remove all mines
    currentPop[1][j]=1; //may be replaced
  }

minesLeft=mines;
replace=0;
next=0;
//Algorithm removes all mines and randomly replaces them one by one.
//If replacement of a mine causes that a route can not be found, that mine is permanently removed.
for k=0 to mines //for all mines
  {
    if(k<>0) //do first iteration with all mines removed, i.e. no mine is replaced
      {
        cut=random(minesLeft); //random cut
        replace=minesAvail[cut][0]; //select random mine to be replaced
        minesLeft=minesLeft-1;
        minesAvail[cut][0]=10000000;
        sortArray(minesAvail,mines,0,TRUE,TRUE); //sort array so that all untried mines move to top

        currentPop[0][replace]=1; //replace mine in currentPop
      }
    **Set up dummyArray according to the population to be evaluated and evaluate each population member
    next=0;
    for j=0 to mines-1
      {
        if(currentPop[0][j]==1) //mine present
          dummyArray[j][2]=1;
        else
          {
            dummyArray[j][2]=0; //mine not present
            next++; //counter that tallies deleted mines
          }
      }
    findRoute();
    fitness=1/((result/bestPath)+next); //fitness=1/(pathLength/bestPath+deletedMines) (larger is better)
                                     //shorter path is favoured, but fewer mines take preference
    if(fitness<0.001) //no route found
      currentPop[0][replace]=0; //has to be removed
    else
      {
        currentPop[0][100]=fitness;
        delMinesTemp=next;
      }
  }
delMines=delMinesTemp;
delMinesText=delMinesText+delMines+", ";
}

procedure GAOptimise()
  {
    integer i,j,k,l,m,p1,p2,next,cut;
    integer startRow, endRow, closestStart, closestEnd;
    real fitness,bestOverallFitness, bestGenFitness, summedGenFitness, randomNum, selectFit;
  }

```



```

delMines=mines;
bestOverallFitness=0; //for this optimisation
bestGenFitness=0; //for this generation
summedGenFitness=0; //total fitnesses summed for this generation

// calculate shortest possible path (height from row containing start node to row containing end node
If(entryLast)
    closestStart=entryLast;
else
    closestStart=entryFirst;
closestEnd=ExitFirst;
startRow=ceil(closestStart/Cols);
endRow=ceil(closestEnd/Cols);
bestPath=realAbs((endRow-startRow)*spacing);

delMinesText="";
//generate the initial population
disposeArray(currentPop); //causes new array to be initialised to 0 including last column (fitness) to 0

if(doSearch) //for search optimisation
{
    makeArray(CurrentPop,searchIterations+2); //array to store the current population of mines
    removed/present
    for i=0 to searchIterations-1
    {
        search();
        if(plot==TRUE) //plot if button was pressed
        {
            plot=FALSE;
            sortArray(currentPop,searchIterations+2,100,FALSE,FALSE); //re-sort array, best to top
            plotBest();
        }
        next=0;
        for j=0 to mines-1
        {
            if(currentPop[0][j]==0) //mine not there
                next++;
            currentPop[i+2][j]=currentPop[0][j]; //copy from row 0 into next row
        }
        currentPop[i+2][100]=currentPop[0][100]; //copy fitness from row 0 into next row
        delMines=next;
    }
    currentPop[0][100]=0;
    sortArray(currentPop,searchIterations+2,100,FALSE,FALSE); //re-sort array, best to top
    next=0;
    for j=0 to mines-1
    {
        if(currentPop[0][j]==0) //for best solution, find removed mines
            next++;
    }
    delMines=next;
    plotBest();
    return;
} //end if(doSearch)

makeArray(CurrentPop,popSize*2); //array to store the current population of mines removed/present
if (doGArandom) //generate initial population for random initialisation
{
    for i=popSize to popSize*2-1 //initial population in bottom half of the array

```

```

    {
    for j=0 to mines-1 //check each mine
    {
    if (randomReal()<initProb)
        currentPop[i][j]=0; //remove mine
    else
        currentPop[i][j]=1; //leave mine
    }
    }
}

if (doGAssearch) //generate initial population for search initialisation
{
    for i=popSize to popSize*2-1 //initial population in bottom half of the array
    {
        search();
        for j=0 to mines-1
            currentPop[i][j]=currentPop[0][j]; //copy valid route to initial population array
        }
        for j=0 to mines
        {
            currentPop[0][j]=0; //clean up array
            currentPop[1][j]=0; //clean up array
        }
    }
}

for k=0 to MaxGenerations //for all generations
{
    if(plot==TRUE) //plot if button was pressed
    {
        plot=FALSE;
        plotBest();
    }
    **Create new generation
    if(k<>0 && Best) //Best mates with all others, for first iteration don't do
    {
        sortArray(currentPop,popSize*2,100,FALSE,FALSE); //re-sort array
        for (i=0; i<popSize;i++) //all in population
        {
            cut=random(mines-2);
            for j=0 to cut //copy first part of genome
            {
                currentPop[popsize+i][j]=currentPop[0][j];
                if(randomReal()<mutate) //mutate
                    if(currentPop[popsize+i][j]==0)
                        currentPop[popsize+i][j]=1;
                else
                    currentPop[popsize+i][j]=0;
            }
            for j=cut+1 to mines-1 //copy second part of genome and exchange
            {
                currentPop[popsize+i][j]=currentPop[i][j];
                if(randomReal()<mutate) //mutate
                    if(currentPop[popsize+i][j]==0)
                        currentPop[popsize+i][j]=1;
                else
                    currentPop[popsize+i][j]=0;
            }
            // currentPop[popsize+i][j]=currentPop[i+1][j];
        }
    }
}
}

```

```

//based on roulette wheel selection
if(k<>0 && Roulette) //not for first generation
{
  if(keepBest) //find fittest between new and old population
    sortArray(currentPop,popSize*2,100,FALSE,FALSE); //re-sort array of old and new populations
  else //only new populations reproduces
  {
    arrayDataMove(currentPop,popSize,popSize,0,TRUE); //move original pop to the top of array
    sortArray(currentPop,popSize,100,FALSE,FALSE); //re-sort array
  }
  for i=0 to (popSize/2)-1
  {
    randomNum=randomReal()*summedGenFitness;
    selectFit=0;
    for p1=0 to popSize-1 //select first parent according to roulette wheel
    {
      selectFit=selectFit+currentPop[p1][100];
      if(selectFit>randomNum)
      {
        break; //p1 is set
      }
    }
    p2=random(popsize); //select second parent at random from previous generation

    **Reproduction
    cut=random(mines-2);
    for j=0 to cut //copy first part of genome from first parent
    {
      currentPop[popsize+i*2][j]=currentPop[p1][j];
      if(randomReal(<mutate) //mutate
      if(currentPop[popsize+i][j]==0)
        currentPop[popsize+i][j]=1;
      else
        currentPop[popsize+i][j]=0;
      currentPop[popsize+i*2+1][j]=currentPop[p2][j];
      if(randomReal(<mutate) //mutate
      if(currentPop[popsize+i+1][j]==0)
        currentPop[popsize+i+1][j]=1;
      else
        currentPop[popsize+i+1][j]=0;
    }
    for j=cut+1 to mines-1 //copy second part of genome and exchange
    {
      currentPop[popsize+i*2][j]=currentPop[p2][j];
      if(randomReal(<mutate) //mutate
      if(currentPop[popsize+i+1][j]==0)
        currentPop[popsize+i+1][j]=1;
      else
        currentPop[popsize+i+1][j]=0;
      currentPop[popsize+i*2+1][j]=currentPop[p1][j];
      if(randomReal(<mutate) //mutate
      if(currentPop[popsize+i][j]==0)
        currentPop[popsize+i][j]=1;
      else
        currentPop[popsize+i][j]=0;
    }
  }
}

bestGenFitness=0; //for this generation

```

```

summedGenFitness=0;
**Set up dummyArray according to the population to be evaluated and evaluate each population member
for i=popSize to popSize*2-1 //for this population, bottom section of array
{
  next=0;
  for j=0 to mines-1
  {
    if(currentPop[i][j]==1) //mine present
      dummyArray[j][2]=1;
    else
    {
      dummyArray[j][2]=0; //mine not present
      next++; //counter that tallies deleted mines
    }
  }
  findRoute();
  fitness=1/((result/bestPath)+next); //fitness=1/(pathLength/bestPath+deletedMines) (larger is better)
                                     //shorter path is favoured, but fewer mines take preference
  summedGenFitness+=fitness;
  if(fitness>bestGenFitness) //for this generation
  {
    bestGenFitness=fitness;
    delMinesOut=next;
  }
  if(fitness>bestOverallFitness)
  {
    bestOverallFitness=fitness;
    delMines=next;
  }
  currentPop[i][100]=fitness; // add fitness to bottom of the sorted array
}
fitOut=bestGenFitness; //best for this generation
genOut=k;
sendMsgToInputs(genOut);
}
sortArray(currentPop,popSize*2,100,FALSE,FALSE); //re-sort array
plotBest();
}

** This messagehandler ensures the route is only calculated when a trigger is received
on triggerIn
{
  integer i, optimise;
// if(triggerIn)
// {
  if(first)
  {
    nodes=rows*cols;

    ** get values if connected
    if(connected)
    {
      sendMsgToAllCons(xStartIn);
      sendMsgToAllCons(yStartIn);
      sendMsgToAllCons(xEndIn);
      sendMsgToAllCons(yEndIn);
    }

    **get array with mine co-ordinates
    if (getPassedArray(minesIn, OrigArray))

```

```
{
  ** The array was passed successfully
}
else  ** FALSE result means the array was not passed yet.
{
  usererror("No Mine array received");
  abort;
}

OrigMines=getDimension(OrigArray);
if(OrigMines>100)
{
  usererror("There may be no more than 100 mines in this simulation");
  abort;
}
**copy x y co-ords to new array
makeArray(dummyArray,origMines);
for i=0 to origMines-1
{
  dummyArray[i][0]=origArray[i][0];
  dummyArray[i][1]=origArray[i][1];
  dummyArray[i][2]=1;
}
//plot mine positions
routeXout=blank;
routeYout=blank;
for i=0 to origMines-1
{
  mineXout=origArray[i][0];
  mineYout=origArray[i][1];
  sendMsgToInputs(MineXout);
  if (allXY)
    sendMsgToInputs(MineYout);
}
mineXout=blank;
mineYout=blank;

findRoute(); //do procedure to find the route

first=false;
xOut=blank;
yOut=blank;
routeXout=blank;
routeYout=blank;
if (gridOn)
{
  for i=1 to nodes
  {
    xOut=nodeXY[i][0];
    yOut=nodeXY[i][1];
    sendMsgToInputs(xOut);
  }
}
if (noRoute)
  GAOptimise();
else
{
  mineXout=blank;
  mineYout=blank;
  **plot the route
```

```
for i=1 to nodes
  {
  if(!shortest[i][3]) //break if a noValue is found
    break;
  pointOut=i;
  RouteXout=nodeXY[shortest[i][3]][0];
  RouteYout=nodeXY[shortest[i][3]][1];
  sendMsgToInputs(RouteXOut);
  if (allXY)
    {
    sendMsgToInputs(routeYOut);
    }
  }
  RouteXout=blank;
  RouteYout=blank;
  xOut=blank;
  yOut=blank;
}
}
```

\*\*Message handlers to prevent excessive message emulation

on xOut

```
{
}
```

on yOut

```
{
}
```

on routexOut

```
{
}
```

on routeyOut

```
{
}
```

on minexOut

```
{
}
```

on mineyOut

```
{
}
```