A Moving Finite Element Approach to Semiconductor Process Modelling in 1-D

John M. Hobbs

September 1993

Submitted to the

Department of Mathematics,

University of Reading,

in partial fulfillment of the requirements for the

Degree of Master of Science

Abstract

The Moving Finite Element Method and modifications to the method are investigated for robustness and efficiency when applied to a problem in Semiconductor Process Modelling. The modifications carried out include the use of Penalty Functions, Gradient Weighting and Graph Massage, and the semiconductor modelling involves the prediction of the dopant concentration within the silicon that constitutes the semiconductor device, so that certain electrical properties of the device can be predicted.

Acknowledgements

I would like to thank Dr.M.J. Baines and Dr.P.K. Sweby of the Mathematics Department of Reading University for their supervision and advice during this work. I would also like to acknowledge SERC for their financial support during the last year.

Chapter 1

Introduction

In order to predict certain electrical properties of a semiconductor device it is necessary to know the distribution of a dopant within the crystalline silicon that constitutes the device, and the process of introducing this dopant and diffusing it within the silicon is known as process modelling. Typically the dopant (eg, arsenic) is introduced into the silicon by ion implantation, through the surface of the silicon, and this results in a high concentration of dopant in a shallow region. The doped silicon is then heated to regrow the damaged crystal which causes the dopant to diffuse within the silicon. This diffusion proces is highly non-linear and gives rise to an interesting problem in numerical modelling.

Unfortunately, the numerical solution of this process is made difficult by certain features that occur during the diffusion. One difficulty arises owing to the formation of a steep front in the concentration profile which is created as the initial peak of the original (implanted) concentration diffuses much quicker than the tail. This front is sharp and advances across the region, so its resolution is of great importance. However, it is also important that the position and structure

of the junction[1] is predicted accurately after diffusion is complete. The position of this junction occurs well into the tail region, which is some five or six orders of magnitude below the initial concentration, and this is a second source of numerical difficulty.

The aim of this dissertation is to formulate a simple model of the diffusion process and to find a method which will solve the problem both accurately and efficiently. To this end the method of moving finite elements (MFE) is used and various modifications to the original MFE method [8] are examined, including the use of penalty functions, weight functions and graph massage[2]. A method for dealing with the second numerical difficulty mentioned above, by means of a transformation, is discussed and the results from the FORTRAN 90 code are presented in Chapter 5. Finally, conclusions are drawn about the various strategies investigated.

[1]Before the dopant is implanted a very small concentration of dopant of opposite sign is distributed uniformly throughout the silicon. The junction is then defined as the place where the concentration of the implanted dopant drops below this background level.

[2]This is a new approach for dealing with the problems of node migration to and from certain regions of the graph and is dealt with in some detail in Chapter 4.

Chapter 2

The Semiconductor Problem

2.1 The Diffusion Model

2.1.1 General Case

It is necessary to start by formulating a model of the non-linear diffusion process that takes place within the crystalline silicon as the dopant diffuses. The details of the diffusion mechanisms involved are omitted here, but they can be found in [3] and [6]. An equation for the conservation of the dopant concentration c is obtained, and can be written in the form

$$\frac{\partial c}{\partial t} = \nabla.(D(c)\nabla c) \tag{2.1}$$

where

$$\left.\begin{aligned} D(c) &= D_i[\tfrac{1+\beta\frac{n_e}{n_i}}{1+\beta}] \\ n_e &= \tfrac{1}{2}[\,c + \sqrt{c^2+4n_i^2}\,] \end{aligned}\right\} \tag{2.2}$$

β, D_i and n_i are constants and the implantation of the dopant is modelled by taking the initial profile to be a Gaussian hump which has a standard deviation

that is small in comparison to the distance over which the the dopant diffuses. Equations (2.1) and (2.2) can be simplified further to obtain the *model* problem

$$\left.\begin{array}{rcl} \frac{\partial c}{\partial t} & = & \nabla.(D(c)\nabla c) \\ D(c) & = & c+\epsilon \end{array}\right\} \tag{2.3}$$

where $\epsilon \ll 1$ (typically $O(10^{-2})$). This new equation has the essential features of equations (2.1) and (2.2) and taking a Gaussian initial profile leads to similar behaviour to that of the full problem.

2.1.2 The 1-D Case

In 1-D, equation (2.3) becomes

$$\frac{\partial c}{\partial t} = \frac{\partial}{\partial x}\left((c+\epsilon)\frac{\partial c}{\partial x}\right) \tag{2.4}$$

and carrying out the differentiation on the RHS gives

$$\frac{\partial c}{\partial t} = \left(\frac{\partial c}{\partial x}\right)^2 + (c+\epsilon)\frac{\partial^2 c}{\partial x^2} \tag{2.5}$$

The boundary conditions are taken to be

$$\left.\begin{array}{rcl} \frac{\partial c}{\partial x} & = & 0 \; at \; x = -a \\ c & \rightarrow & 0 \; as \; |x| \rightarrow \infty \end{array}\right\} \tag{2.6}$$

and the initial condition is a Gaussian hump, perpendicular to the surface of the silicon, given by

$$c = \lambda e^{\frac{-(x-\omega)^2}{2\sigma^2}} \; at \; t = 0 \tag{2.7}$$

where

$x = -a$ = silicon surface,

λ = initial height of Gaussian,

ω = centre of Gaussian,

σ = standard deviation of Gaussian.

A typical profile for the initial concentration can be seen in Figure 1.1.

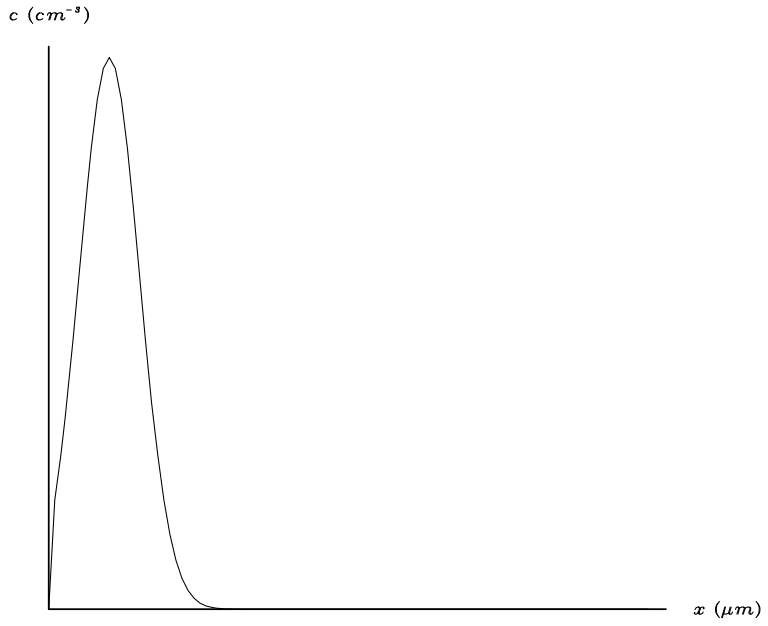

Figure 1.1 Initial dopant concentration profile.

Clearly, the second boundary condition in (2.6) cannot be explicitly applied in a numerical sense, so a Neumann BC, $\frac{\partial c}{\partial x} = 0$, is applied at $x = b$ where b is fixed and large enough so that the junction will lie inside [$-a$, b] after diffusion is complete. Alternatively, a moving boundary could be used, but the study of this boundary is not considered here. More details of moving boundaries and the MFE method can be found in [9].

2.2 A Transformation

2.2.1 Reasons for a Transformation

Detailed information needs to be provided in the tail region in order to accurately resolve the position of the junction which forms there. However, using the dopant concentration c as the variable can cause problems because it is very small in magnitude and hard to resolve. As a result of this, finding a point in the tail where the concentration has a particular value is prone to error. Therefore, it is argued that the dopant concentration may not be the best choice of variable from a physical point of view and that a variable which is related more directly to the position of the junction should be used.

2.2.2 The Transformation

This leads to the transformation, suggested by Please and Sweby [10], which stretches the ordinate in the tail region. The moving front in the solution profile has a near-shock structure, so it is argued that a more attractive variable would be the profile speed in the x direction, V. To relate this to c it is observed that flux is swept out by a near-shock at a rate cV and that this is balanced by the flux rate $D(c)\nabla c$ in the non-linear diffusion mechanism. Then ϕ, the velocity potential of V is introduced, and a comparison of the flux rates gives

$$c\nabla\phi = D(c)\nabla c \tag{2.8}$$

which in turn gives

$$\phi = \int \frac{D(c)}{c} dc \tag{2.9}$$

Using equation (2.9) with the model problem (2.3), where $D(c) = c + \epsilon$, gives the new variable

$$\phi = c + \epsilon \, log_e(c) \tag{2.10}$$

For small c, $\phi \sim \epsilon \, log_c(c)$, so c has been stretched logarithmically in this region (and only in this region), which allows greater resolution in the tail where the solution is no longer exponential but quadratic (since the initial function is quadratic for small c under this transformation). For large c, $\phi \sim c$, so the behaviour of the solution is basically unaltered in this region and this is needed to model the early dominance of the diffusion mechanism.

2.2.3 Application to the Semiconductor Problem

Substituting equation (2.8) into equation (2.1) gives

$$\frac{\partial c}{\partial t} = \nabla.(c \nabla \phi) \tag{2.11}$$

and differentiating (2.10) with respect to t gives

$$\frac{\partial \phi}{\partial t} = \frac{D(c)}{c} \frac{\partial c}{\partial t} \tag{2.12}$$

Substituting equation (2.11) into equation (2.12) gives

$$\begin{aligned} \frac{\partial \phi}{\partial t} &= \frac{D(c)}{c} \nabla.(c \nabla \phi) \\ &= \frac{D(c)}{c} \left[\nabla c \nabla \phi + c \nabla^2 \phi \right] \\ &= \frac{D(c)}{c} \nabla c \nabla \phi + D(c) \nabla^2 \phi \end{aligned} \tag{2.13}$$

Rearranging equation (2.8) and substituting into equation (2.13) gives

$$\begin{aligned} \frac{\partial \phi}{\partial t} &= \nabla \phi \nabla \phi + D(c) \nabla^2 \phi \\ &= (\nabla \phi)^2 + D(c) \nabla^2 \phi \end{aligned} \tag{2.14}$$

In 1-D equation (2.14) becomes

$$\frac{\partial \phi}{\partial t} = \left(\frac{\partial \phi}{\partial x} \right)^2 + D(c) \frac{\partial^2 \phi}{\partial x^2} \tag{2.15}$$

and this does not look very different to equation (2.5). However, the advantage of equation (2.15) is the scale of ϕ when compared to that of c, and the severity of the numerical difficulties is greatly reduced.

Although c appears in equation (2.15) which is a PDE for ϕ, it is a simple task to calculate c from ϕ using (2.10). The FORTRAN 90 code uses Newton iteration to carry out this inversion which is computationally inexpensive: typically only three or four iterations are required, although more iterations are required at certain values[1] of c.

It is worth noting that for the full problem, equations (2.1), (2.2) and (2.9) imply that a much more complicated relationship than (2.10) exists between the variables c and ϕ, but the model diffusion coefficient of (2.3), namely $D(c) = c + \epsilon$, gives the qualitative scaling that is required. Hence, it is sufficient for this study to use (2.10) and avoid the complications of using $D(c)$ as in (2.2).

[1]These *critical* values are around the point where the initial approximation changes from $c = \phi$ to $c = e^{\frac{\phi}{\epsilon}}$, but even at these values the Newton solver takes only six or seven iterations to converge.

Chapter 3

The Finite Element Method

3.1 Reasons for Using Finite Elememts

In Chapter 1 it was stated that one of the aims of the dissertation was to find a method that is able to accurately track and model the position of the moving front. On a fixed grid it would be necessary to have a very fine grid to provide a good resolution of the moving front, but away from the front the solution is smooth and the fine grid is not necessary. To avoid the computational inefficiency of using a large number of grid-points, the finite element method (FEM) is used since it is able to provide the irregular grid that is needed for this problem at any fixed time. Furthermore, if the grid is allowed to move then it is hoped that the 'correct' concentration of nodes will be provided at the required positions.

3.2 The Fixed Finite Element Method

Consider the general PDE

$$\frac{\partial c}{\partial t} = \mathcal{L}\, c \tag{3.1}$$

where $\mathcal{L}$ is a spatial differential operator which involves, at most, second order derivatives. A variational principle can be obtained for equation (3.1) by considering the weighted L_2 norm of the PDE which is

$$R = \int_a^b \left(\frac{\partial c}{\partial t} - \mathcal{L}\, c \right)^2 w \, dc \tag{3.2}$$

where w is a weight function to be chosen later. If R is minimised with respect to $\frac{\partial c}{\partial t}$ (by setting $\frac{\partial R}{\partial c_t} = 0$) then equation (3.1) is recovered. This procedure can be extended to the case where c is replaced by an approximation and an intuitive interpretation is that R measures the residual, ie the degree to which the original equation fails to be satisfied.

The standard Galerkin method of finite elements is derived by approximating c by

$$c\,(x\,,t\,) = \sum_{j=1}^{n} c_j(t)\,\alpha_j(x) \tag{3.3}$$

where

$c_j(t)$ are the time-dependent amplitudes,

$\alpha_j(x)$ are the time-independent basis functions.

and the time derivative of c, from equation (3.3), is given by

$$\frac{\partial c}{\partial t} = \sum_{j=1}^{n} \dot{c}_j(t)\,\alpha_j(x) \tag{3.4}$$

Discretised equations for the nodal amplitudes can be found by minimising R with respect to the time derivatives of the amplitudes, ie by setting $\frac{\partial R}{\partial \dot{c}_j} = 0$. This generates a set of equations for the c_j which give the *'best fit'* of $\frac{\partial c}{\partial t}$ to $\mathcal{L}\, c$ and these are given explicitly by

$$\langle \alpha_i\,,\ \frac{\partial c}{\partial t} - \mathcal{L}\, c \rangle = 0\,, \quad i = \{1, \ldots, n\}$$

$$\Rightarrow \langle \alpha_i \, , \, \sum_{j=1}^{n} \dot{c}_j \, \alpha_j - \mathcal{L}\, c \rangle = 0 \, , \quad i = \{1, \ldots, n\}$$

$$\Rightarrow \langle \alpha_i \, , \, \sum_{j=1}^{n} \dot{c}_j \, \alpha_j \rangle = \langle \alpha_i \, , \, \mathcal{L}\, c \rangle \, , \quad i = \{1, \ldots, n\}$$

$$\Rightarrow \sum_{j=1}^{n} \langle \alpha_i \, , \, \alpha_j \rangle \, \dot{c}_j = \langle \alpha_i \, , \, \mathcal{L}\, c \rangle \, , \quad i = \{1, \ldots, n\} \tag{3.5}$$

where

$$\langle u \, , \, v \rangle = \int_a^b u \, v \, w \, dc$$

Equation (3.5) represents a system of n ODEs in n unknowns (the nodal amplitudes), so the original PDE problem has been reduced to an ODE system and the integration of these ODEs gives a continuously evolving function *c(t)* which approximates the solution of (3.1).

3.3 Basis Functions

Conventionally, finite element basis functions are chosen so that they vanish outside of a fixed interval, which results in the matrix given by the inner products $\langle \alpha_i \, , \, \alpha_j \rangle$ in equation (3.5) being banded or sparse, since distant basis functions do not overlap. This simplifies the task of evaluating the inner products (and inverting the matrix if necessary). The most common finite elements that are used are *piecewise linear* finite elements and it is these that are used in the following work.

A grid of points, s_j is defined such that $a = s_0 < \ldots < s_j < \ldots < s_n = b$. Then each basis function, α_j, is defined to be 1 at s_j, linear in the interval $[s_{j-1} \, , \, s_{j+1}]$ and 0 outside the interval $[s_{j-1} \, , \, s_{j+1}]$. More explicitly, the basis functions are given by

$$\alpha_j = \begin{cases} \frac{x - s_{j-1}}{s_j - s_{j-1}} & x \in [s_{j-1}\,,\, s_j] \\ \frac{s_{j+1} - x}{s_{j+1} - s_j} & x \in [s_j\,,\, s_{j+1}] \\ 0 & otherwise \end{cases} \tag{3.6}$$

and a typical basis function α_j can be seen in Figure 3.1. With this choice of

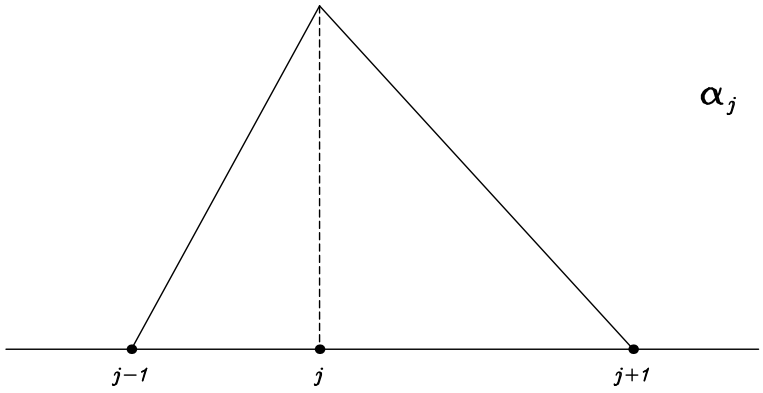

Figure 3.1

basis functions the matrix resulting from the inner products of equation (3.5) is tri-diagonal.

3.4 Problems of the Fixed FEM

While the FEM discussed above can provide a non-uniform grid and thus reduce the number of nodes by concentrating nodes in regions where the solution gradient varies rapidly and placing fewer nodes in regions where the solution is smooth, it cannot adapt itself to a problem where these regions move in time. In order to track these moving regions it is necessary to allow the grid to adapt as the solution evolves. From the point of view of the FEM it is desired to find a way in which the basis functions can adapt themselves to the evolving solution. It is

for this reason that the method of moving finite elements (MFE) was introduced and it is this method that is examined here in some detail.

3.5 The Moving Finite Element Method

The MFE method is an extension of the fixed FEM where the nodes are allowed to move over time, and this is an example of a dynamic rezoning strategy. The grid positions s_j become functions of time and c is now approximated by

$$c(x\,,\,t) = \sum_{j=1}^{n} c_j(t)\,\alpha_j(x\,,\,s_j(t)) \tag{3.7}$$

The time derivative of c is much more complicated than for the fixed FEM owing to the dependence of α_j on the nodes s_{j-1}, s_j and s_{j+1}, which are all free to move in time.

3.5.1 Derivation of $\frac{\partial c}{\partial t}$ for the MFE

Differentiating equation (3.7) with respect to t gives

$$\frac{\partial c}{\partial t} = \sum_{j=1}^{n} \left[\dot{c}_j(t)\,\alpha_j(x\,,\,s_j(t)) + c_j(t)\,\frac{\partial}{\partial t}\alpha_j(x\,,\,s_j(t)) \right] \tag{3.8}$$

Now,

$$\begin{aligned} \sum_{j=1}^{n} c_j(t)\,\frac{\partial}{\partial t}\alpha_j(x\,,\,s_j(t)) &= \sum_{j=1}^{n} c_j \frac{\partial \alpha_j}{\partial \mathbf{s}} \frac{d\mathbf{s}}{dt} \\ &= \sum_{j=1}^{n} c_j \sum_{i=1}^{n} \frac{\partial \alpha_j}{\partial s_i}\dot{s}_i \end{aligned} \tag{3.9}$$

Consider a general basis function α_j for a piecewise linear approximation. Clearly, $\frac{\partial \alpha_j}{\partial s_i}$ is only non-zero for $s_i \in \{s_{j-1},\, s_j,\, s_{j+1}\}$.

So,

$$\left.\begin{array}{rcl} \frac{\partial \alpha_j}{\partial s_{j-1}}\dot{s}_{j-1} & = & -\frac{(s_j - x)}{(s_j - s_{j-1})^2}\dot{s}_{j-1} \\ \frac{\partial \alpha_j}{\partial s_j}\dot{s}_j & = & \left(-\frac{(x - s_{j-1})}{(s_j - s_{j-1})^2} + \frac{(s_{j+1} - x)}{(s_{j+1} - s_j)^2}\right)\dot{s}_j \\ \frac{\partial \alpha_j}{\partial s_{j+1}}\dot{s}_{j+1} & = & \frac{(x - s_j)}{(s_{j+1} - s_j)^2}\dot{s}_{j+1} \end{array}\right\} \tag{3.10}$$

Hence,

$$\sum_{j=1}^{n} c_j \sum_{i=1}^{n} \frac{\partial \alpha_j}{\partial s_i}\dot{s}_i = \sum_{j=1}^{n} [\, \textit{Sum of the RHS of equations (3.10)}\,] \tag{3.11}$$

Collect the $\dot{s}_j$ terms, giving

$$\begin{array}{rl} -\dfrac{(s_{j+1} - x)}{(s_{j+1} - s_j)^2} c_{j+1}\dot{s}_j & \textit{from} \quad \dfrac{\partial \alpha_{j+1}}{\partial s_j} \\ \left(-\dfrac{(x - s_{j-1})}{(s_j - s_{j-1})^2} + \dfrac{(s_{j+1} - x)}{(s_{j+1} - s_j)^2}\right) c_j\dot{s}_j & \textit{from} \quad \dfrac{\partial \alpha_j}{\partial s_j} \\ \dfrac{(x - s_j)}{(s_j - s_{j-1})^2} c_{j-1}\dot{s}_j & \textit{from} \quad \dfrac{\partial \alpha_{j-1}}{\partial s_j} \end{array}$$

and then sum them to get

$$\left[-\frac{(s_{j+1} - x)}{(s_{j+1} - s_j)}\frac{(c_{j+1} - c_j)}{(s_{j+1} - s_j)} - \frac{(x - s_{j-1})}{(s_j - s_{j-1})}\frac{(c_j - c_{j-1})}{(s_j - s_{j-1})}\right]\dot{s}_j = \beta_j\dot{s}_j \tag{3.12}$$

say, where

$$\beta_j = \begin{cases} -m_j\,\alpha_j & x \in [s_{j-1}\,,\, s_j] \\ -m_{j+1}\,\alpha_j & x \in [s_j\,,\, s_{j+1}] \\ 0 & \textit{otherwise} \end{cases} \tag{3.13}$$

and

$$m_j = \frac{(c_j - c_{j-1})}{(s_j - s_{j-1})} \tag{3.14}$$

Then, equations (3.11) and (3.12) give

$$\sum_{j=1}^{n} c_j \frac{\partial \alpha_j}{\partial \mathbf{s}}\frac{d\mathbf{s}}{dt} = \sum_{j=1}^{n} \beta_j\dot{s}_j \tag{3.15}$$

Finally, substituting equation (3.15) into equation (3.8) gives

$$\frac{\partial c}{\partial t} = \sum_{j=1}^{n} [\alpha_j\dot{c}_j + \beta_j\dot{s}_j] \tag{3.16}$$

The β_j's will be discontinuous (since the m_j's can be regarded as a discontinuous function) and a typical basis function β_j can be seen in Figure 3.2.

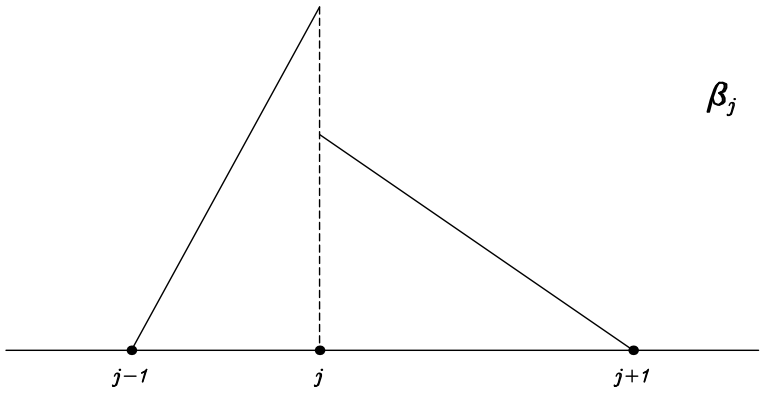

Figure 3.2

3.5.2 Minimisation of R

If the residual, R, in equation (3.2) is now minimised with respect to $\dot{c}_j$ and $\dot{s}_j$, by setting $\frac{\partial R}{\partial \dot{c}_j} = 0 = \frac{\partial R}{\partial \dot{s}_j}$, then this generates movements which give the *'best fit'*, in the same sense as before, over the enlarged space of nodal amplitudes and nodal velocities, and therefore, it is claimed, a better fit, since the grid moves in such a way as to minimise the residual and hopefully this will optimise the solution. The validity of this claim is somewhat questionable though, as the node movement may have more to do with the 'crudeness' of using linear basis functions than anything else, and details of this can be found in [1].

The result of the minimisation of R in this way is referred to as a method of lines (MOL) and the equations are given more explicitly by

$$\left.\begin{array}{rcl} \langle \alpha_i \,,\, \frac{\partial c}{\partial t} - \mathcal{L}\, c \rangle & = & 0 \\ \langle \beta_i \,,\, \frac{\partial c}{\partial t} - \mathcal{L}\, c \rangle & = & 0 \end{array}\right\} \quad i = \{1, \ldots, n\}$$

$$\Rightarrow \left.\begin{array}{rcl} \langle \alpha_i \, , \, \sum_{j=1}^{n} [\alpha_j \, \dot{c}_j + \beta_j \, \dot{s}_j] - \mathcal{L} \, c \rangle & = & 0 \\ \langle \beta_i \, , \, \sum_{j=1}^{n} [\alpha_j \, \dot{c}_j + \beta_j \, \dot{s}_j] - \mathcal{L} \, c \rangle & = & 0 \end{array}\right\} \quad i = \{1, \ldots, n\}$$

$$\Rightarrow \left.\begin{array}{rcl} \sum_{j=1}^{n} \langle \alpha_i \, , \, \alpha_j \rangle \dot{c}_j + \sum_{j=1}^{n} \langle \alpha_i \, , \, \beta_j \rangle \dot{s}_j & = & \langle \alpha_i \, , \, \mathcal{L} \, c \rangle \\ \sum_{j=1}^{n} \langle \beta_i \, , \, \alpha_j \rangle \dot{c}_j + \sum_{j=1}^{n} \langle \beta_i \, , \, \beta_j \rangle \dot{s}_j & = & \langle \beta_i \, , \, \mathcal{L} \, c \rangle \end{array}\right\} \quad i = \{1, \ldots, n\} \quad (3.17)$$

Equations (3.17) represent a system of $2\,n$ ODEs in $2\,n$ unknowns and the matrix associated with the inner products of (3.17) is $2 \, \mathrm{x} \, 2$ block tri-diagonal.

3.6 Structure of the MFE Matrix

If $\mathbf{A}$ is the matrix associated with the inner products of (3.17) then each $2 \, \mathrm{x} \, 2$ block, $\mathbf{B}_{ij}$ ($i = \{1, \ldots, n\}$), of $\mathbf{A}$ is given by

$$\mathbf{B}_{ij} = \left[\begin{array}{cc} \langle \alpha_i \, , \, \alpha_j \rangle & \langle \alpha_i \, , \, \beta_j \rangle \\ \langle \beta_i \, , \, \alpha_j \rangle & \langle \beta_i \, , \, \beta_j \rangle \end{array} \right] , \quad j = \{1, \ldots, n\} \quad (3.18)$$

It is a simple exercise to show that $\langle \alpha_i \, , \, \beta_j \rangle \equiv \langle \beta_i \, , \, \alpha_j \rangle$ and hence, that each block is symmetric. It is also a straightforward exercise to show that the super-diagonal block $i = k$ is identical to the sub-diagonal block $i = k{+}1$ and thus the full matrix is symmetric. Therefore, it is only necessary to calculate the inner products for the diagonal blocks and the super-diagonal (or sub-diagonal) blocks to determine the matrix and these are given explicitly by :

Diagonal entries

$$\left.\begin{array}{rcl} \langle \alpha_i \, , \, \alpha_j \rangle \; \rightarrow \; \langle \alpha_j \, , \, \alpha_j \rangle & = & \frac{1}{3} [\Delta s_j + \Delta s_{j+1}] \\ \langle \alpha_i \, , \, \beta_j \rangle \; \rightarrow \; \langle \alpha_j \, , \, \beta_j \rangle & = & \frac{1}{3} [\Delta c_j + \Delta c_{j+1}] \\ \langle \beta_i \, , \, \beta_j \rangle \; \rightarrow \; \langle \beta_j \, , \, \beta_j \rangle & = & \frac{1}{3} [\Delta c_j m_j + \Delta c_{j+1} m_{j+1}] \end{array}\right\} \quad j = \{1, \ldots, n\} \quad (3.19)$$

Super-diagonal entries

$$\left.\begin{array}{rcl} \langle \alpha_i \, , \, \alpha_j \rangle & \rightarrow & \langle \alpha_{j+1} \, , \, \alpha_j \rangle \; = \; \frac{1}{6} \Delta s_{j+1} \\ \langle \alpha_i \, , \, \beta_j \rangle & \rightarrow & \langle \alpha_{j+1} \, , \, \beta_j \rangle \; = \; -\frac{1}{6} \Delta c_{j+1} \\ \langle \beta_i \, , \, \beta_j \rangle & \rightarrow & \langle \beta_{j+1} \, , \, \beta_j \rangle \; = \; \frac{1}{6} \Delta c_{j+1} m_{j+1} \end{array}\right\} \quad j = \{1, \ldots, n\} \tag{3.20}$$

where

$$\begin{aligned} \Delta s_j &= \text{Distance between } s_j \text{ and } s_{j-1}, \\ \Delta c_j &= \text{Difference in height between } c_j \text{ and } c_{j-1}. \end{aligned}$$

The RHS vector, $\mathbf{g}$, associated with the inner products of equation (3.17) is given by

$$\mathbf{g}_i = \left[\begin{array}{c} \langle \alpha_i \, , \, \mathcal{L}\, c \rangle \\ \\ \langle \beta_i \, , \, \mathcal{L}\, c \rangle \end{array} \right] \, , \quad i = \{1, \ldots, n\} \tag{3.21}$$

and owing to the general nature of $\mathcal{L}\, c$ it is necessary to calculate these inner products using some form of numerical integration. The FORTRAN 90 code uses 8-point Gaussian quadrature to numerically integrate these terms and this seems to provide enough accuracy for the MFE method. (Using a greater number of sampling points did not significantly improve the approximation.)

The vector of unknowns, $\dot{\mathbf{y}}$, is given by

$$\dot{\mathbf{y}} = [\ldots, \dot{c}_i, \dot{s}_i, \ldots]^T \, , \quad i = \{1, \ldots, n\} \tag{3.22}$$

Hence, the MFE system which is given by equations (3.18), (3.21) and (3.22) is

$$\mathbf{A}\, \dot{\mathbf{y}} = \mathbf{g} \tag{3.23}$$

which is a semi-discrete ODE system that needs to be numerically integrated in time to obtain the fully discrete system.

3.7 Time-stepping and Conjugate Gradients

3.7.1 Numerical Time-stepping

To advance the time-step numerically a finite difference time-stepping scheme is introduced and the FORTRAN 90 code uses an explicit Euler method to achieve this. The time-step is automatically chosen by the program to take the largest time-step possible without causing the nodes to merge and while remaining within certain limits (Δt_{min} , Δt_{max}). The user is free to choose the upper time limit, but some sense should be applied and a limit which is 'too large' should be avoided. If any of the nodes should merge in a time which is less than Δt_{max}, then half of the minimum merging time of all the nodes which merge is taken, or the minimum time-step Δt_{min} if this is larger. It was found that the nodes did not merge for any of the runs carried out for the semiconductor problem.

3.7.2 Solution of $\mathbf{A}\dot{\mathbf{y}} = \mathbf{g}$

Since piecewise linear basis functions have been used, a pre-conditioned conjugate gradient (PCG) algorithm was used to invert the symmetric mass matrix $\mathbf{A}$. If $\mathbf{D}$ is taken to be the matrix consisting of the diagonal blocks of $\mathbf{A}$, then pre-conditioning the system (3.23) by $\mathbf{D}^{-1}$ allows the system to be solved very efficiently, requiring only two or three iterations [2], and for the semiconductor problem one iteration[1] was sufficient for a majority of the time-steps. The only exception to this occurred when penalty functions were used (see the next section) and the number of iterations went up to between[2] 10 and 30.

[1] A tolerance of 10^{-8} was used to test the difference between succesive iterations.
[2] The number of iterations depends on the value of the parameter δ.

3.8 Parallelism

Unfortunately, the MFE method is not without its problems. One such problem is that of *parallelism* and this occurs when the equations in the system (3.23) become singular (or nearly singular) at any point. This is a problem because the *'best fit'* is now ambiguous and this creates difficulties for the ODE solver. A singularity occurs when the solutions on adjacent cells lie on the same line, but it is possible to move the offending node without affecting the MFE approximation. Thus, the singularity can be removed by specifying any consistent value for the motion of the node.

3.8.1 Penalty Functions

An alternative approach to dealing with parallelism is to use *penalty functions* which have the effect of adding a small positive definite term

$$R_{+ve} = \epsilon \sum_{j=1}^{n} \frac{(\dot{s}_j - \dot{s}_{j-1})^2}{(s_j - s_{j-1})} \tag{3.24}$$

to the residual R in (3.2), where R_{+ve} only depends on the nodal positions and their time derivatives [4]. The new residual becomes

$$R_p = R + R_{+ve} = \int_a^b \left(\frac{\partial c}{\partial t} - \mathcal{L}\, c \right)^2 w \, dx + \epsilon \sum_{j=1}^{n} \frac{(\dot{s}_j - \dot{s}_{j-1})^2}{(s_j - s_{j-1})} \tag{3.25}$$

and applying the condition $\frac{\partial R_p}{\partial \dot{s}_j} = 0 = \frac{\partial R_p}{\partial \dot{c}_j}$ gives rise to some new equations from the $\frac{\partial R_p}{\partial \dot{s}_j} = 0$ condition, but not from the $\frac{\partial R_p}{\partial \dot{c}_j} = 0$ condition, since R_{+ve} does not depend explicitly on $\dot{c}_j$.

3.8.2 Changes to the Matrix A

Consider $\frac{\partial R_{+ve}}{\partial \dot{s}_j} = 0$. Then for any j,

$$\frac{\partial}{\partial \dot{s}_j} \epsilon \sum_{j=1}^{n} \frac{(\dot{s}_j - \dot{s}_{j-1})^2}{(s_j - s_{j-1})} = 0$$

$$\Rightarrow \epsilon \left[\frac{(\dot{s}_j - \dot{s}_{j-1})}{(s_j - s_{j-1})} - \frac{(\dot{s}_{j+1} - \dot{s}_j)}{(s_{j+1} - s_j)} \right] = 0$$

$$\Rightarrow \epsilon \left[-\frac{1}{(s_j - s_{j-1})} \dot{s}_{j-1} + \left(\frac{1}{(s_j - s_{j-1})} + \frac{1}{(s_{j+1} - s_j)} \right) \dot{s}_j - \frac{1}{(s_{j+1} - s_j)} \dot{s}_{j+1} \right] = 0$$

So, minimising the new residual (3.25) with respect to $\dot{s}_j$ and $\dot{c}_j$ has the effect of adding small positive definite terms to the matrix **A** in (3.17) and the changes to this matrix are given by:

Changes on diagonal block

$$\langle \beta_i \, , \, \beta_j \rangle \rightarrow \langle \beta_j \, , \, \beta_j \rangle + \epsilon \left[\frac{1}{(s_j - s_{j-1})} + \frac{1}{(s_{j+1} - s_j)} \right] , \quad i = \{1, \ldots, n\}$$

Changes on super-diagonal block

$$\langle \beta_i \, , \, \beta_j \rangle \rightarrow \langle \beta_{j+1} \, , \, \beta_j \rangle - \frac{\epsilon}{(s_{j+1} - s_j)} , \quad i = \{1, \ldots, n\}$$

The effect of adding this *internodal viscosity* is to restrain the node motion in regions where the usual restoring force on a node becomes small and the optimum choice of the constant ϵ is given by Miller to be ten times the square of the desired truncation error. This should be large enough to avoid numerical difficulties, but small enough so that grid motion will not be affected in regions where the original matrix is not nearly singular.

The FORTRAN 90 code makes use of both of these procedures to deal with the problem of parallelism, but as parallelism wasn't a difficulty that was encountered with the semiconductor problem it is hard to assess the usefulness of the penalty functions.

3.9 Node Tangling

While the MFE method is expected to cause the grid to concentrate in regions of steep gradient, there is also a tendency for most (or all) of the nodes to concentrate in the region of the moving front, leaving the rest of the region with very few (or no) nodes. Even worse is the case where nodes overtake each other causing a tangling of the grid, but if a "sensible" time-step is taken then this shouldn't happen. To prevent the possibility of this happening and to allow nodes to remain in smooth regions away from the front, it is necessary to reduce the influence of regions of steep gradient. This is achieved by taking the weight function w, in equation (3.2), to be

$$w = (1 + m_j^2)^{-\frac{1}{2}}, \quad j = \{1, \ldots, n\} \tag{3.26}$$

where m_j is the approximate derivative to $\frac{\partial c}{\partial x}$ and is given in (3.14). The effect of this weight function in (3.2) is to de-emphasise regions of the graph where the gradient is large and to convert the integral over x to an integral over the arc-length of the solution curve. It is hoped that the use of the weight function w will result in better nodal movement.

3.10 Conservation

Since the equations of the original semiconductor problem (2.1) and (2.2) were based on a conservation law the extent to which the MFE method preserves the exact conservation properties in the discretised equations should be considered. Glasser [4] shows that the MFE method preserves the exact conservation property if the weight functions are chosen to be constant. However, a constant

weight function does not have the ability to de-emphasize regions of the graph as required and a non-uniform weight function destroys the exact conservation property. Therefore, a decision has to be made as to which property is 'least' desired, and if a non-uniform weight function is used, then it is hoped that the additional accuracy arising from an improved nodal distribution will offset the loss of the exact conservation property.

3.11 Recovery

When solving equations (3.17) it is necessary to evaluate the inner product

$$\langle \beta_i \, , \, \mathcal{L}\, c \rangle \tag{3.27}$$

However, since $\mathcal{L}$ contains second derivatives which don't exist and the β_i are discontinuous[3], this inner product does not exist and it needs to replaced by

$$\langle \beta_i \, , \, \mathcal{L}\, (\mathcal{S}\, c) \rangle \tag{3.28}$$

where $\mathcal{S}$ is a smoothing operator defined explicitly as $\mathcal{S}\, c(x) = v(x)$ [5] by constructing a *recovered* function $v(x)$ from the piecewise linear approximation to c, or its gradient $\frac{\partial c}{\partial x}$, which has sufficient continuity to allow the evaluation of (3.28). Suitable functions $v(x)$ may be constructed by fitting local polynomials of sufficiently high order to c or $\frac{\partial c}{\partial x}$.

It can be shown that taking a Hermite cubic interpolation between nodes i and $i-1$ is equivalent to δ-mollification (ie, arbitrary smoothing of c) and since $v(x)$ can be defined in a variety of other ways as well, the recovery procedure

[3] Usually, integration by parts copes with the problems of second derivatives in finite element approximation, but the discontinuity in the β_i prevents this.

offers greater flexibility than the other methods with no loss of computational efficiency[4]. The use of recovery can also improve accuracy and allow larger time-steps and it is for these reasons that the FORTRAN 90 code uses the recovery procedure.

3.11.1 Choice of Recovery Function

A quadratic function $\frac{\partial v}{\partial x}$ is fitted to the MFE gradient $\frac{\partial c}{\partial x}$ to exploit ideas of superconvergence [5], and this function is defined on the element between nodes i and $i-1$ by

$$\left.\begin{aligned} \tfrac{\partial v}{\partial x}(s_{i-1}) &= (1-\theta)\, m_{i-1} + \theta\, m_i \\ \tfrac{\partial v}{\partial x}(\tfrac{1}{2}(s_{i-1}+s_i)) &= m_i \\ \tfrac{\partial v}{\partial x}(s_i) &= \psi\, m_i + (1-\psi)\, m_{i+1} \end{aligned}\right\}$$

where

$$\begin{aligned} \theta &= \tfrac{\Delta s_i}{(\Delta s_i + \Delta s_{i-1})} \\ \psi &= \tfrac{\Delta s_i}{(\Delta s_i + \Delta s_{i+1})} \end{aligned}$$

For the semiconductor problem

$$\begin{aligned} \mathcal{L}\, c &= \frac{\partial}{\partial x}\left(D(c)\,\frac{\partial c}{\partial x}\right) \\ \rightarrow \mathcal{L}\,(\mathcal{S}\, c) &= \frac{\partial}{\partial x}\left(D(\mathcal{S}\, c)\,\frac{\partial(\mathcal{S}\, c)}{\partial x}\right) \\ \rightarrow \mathcal{L}\, v &= \frac{\partial}{\partial x}\left(D(v)\,\frac{\partial v}{\partial x}\right) \\ &= \frac{\partial}{\partial x}(D(v))\frac{\partial v}{\partial x} + \frac{\partial}{\partial x}\left(\frac{\partial v}{\partial x}\right) \\ &= \frac{\partial}{\partial x}\left(D(v)\right)\frac{\partial v}{\partial x} + \frac{\partial^2 v}{\partial x^2} \end{aligned} \tag{3.29}$$

[4] This is only true in the 1-D case. In 2-D, the recovery procedure becomes a lot more complicated and may not be the best way of evaluating the inner products.

and it should be noted that in equation (3.29) all of the terms which involve the smoothing operator and are differentiated with respect to x need to be recovered, namely $\frac{\partial}{\partial x}(D(v))$ and $\frac{\partial^2 v}{\partial x^2}$. While the recovery of $\frac{\partial}{\partial x}(D(v))$ is not *strictly* necessary, it does help with the nodal velocities and should be used in practice as better numerical results can be obtained.

Chapter 4

Graph Massage

4.1 The Need For Graph Massage

While the instantaneous movement of the nodes using the gradient weighted MFE (GWMFE) method are optimal with respect to the minimisation of the residual R, it is sometimes noted that node depletion and excessively long line segments can gradually occur in certain regions of the solution graph over 'long' periods of time. Conversely, moving fronts can push too many nodes into small regions of the solution graph over 'long' periods of time. To remedy this problem a technique has been developed by Andrew Kuprat which assesses the GWMFE solution graph at regular intervals and adds or deletes nodes where appropriate. This technique is called *Graph Massage* [7]. Owing to the continuous movement of the nodes it is expected that the solution graph will be massaged fairly infrequently and then, that only a small number of nodes will be added or deleted at any time.

The graph massage algorithm is a FORTRAN 90 subroutine which creates nodes according to two criteria (*break* and *length*) and annihilates nodes using one

criteria (*break*). Each of these creation and annihilation criteria will be referred to as a *module*.

4.2 Properties of Graph Massage

The graph massage algorithm is required to satisfy certain properties and these are as follows.

4.2.1 Inter-Modular Stability (IMS)

If the graph massage algorithm was naive then there would be a possibilty that infinite loops of the following type could occur

1) The annihilation module decides that a given node is not needed and deletes it.
2) One of the two creation modules decides that the resulting region of the graph is 'node depleted' and adds a new node.
3) The annihilation module decides that the new node is not needed and deletes it.

etc.

This sort of pointless interaction needs to be avoided and this requirement is called *Inter-Modular Stability*.

IMS is achieved by allowing the creation modules to be 'dumb' and making the annihilation module 'intelligent'. This means that the two creation modules are allowed to add nodes if their criteria for node creation is satisfied, but the annihilation module can only delete a node if the resultant graph in that region does not require the creation modules to add nodes.

4.2.2 Perturbational Stability (PS)

Assume that graph massage has been performed on a graph and that the resultant graph is G with an associated solution vector $\mathbf{c}$. Now suppose that G is perturbed to a new graph G' with the corresponding solution vector $\mathbf{c}' = \mathbf{c} + \boldsymbol{\epsilon}$. Then graph massage should refuse to alter G' in any way provided that $\boldsymbol{\epsilon}$ is sufficiently small and this is called *Perturbational Stability*.

This stability is important when $\mathbf{c}$ represents the GWMFE approximation to a time-dependent PDE, because the graph massage routine will be called every few time-steps and small perturbations in $\mathbf{c}$ will occur between each call owing to the evolutionary nature of the PDE. Therefore, to avoid computational inefficiencies arising from changes made to the graph, it is desired that graph massage will only make changes to G occasionally.

In practice, perturbational stability is achieved by working with two sets of tolerances. After the user has input various tolerances, graph massage increases or decreases each tolerance by a user-specified stability factor so that graph massage is discouraged. If, however, creation or annihilation is required with these 'harder' *trigger* tolerances, then all of the tolerances are relaxed to their original values and the graph massage modules are called. The output solution vector $\mathbf{c}$ now satisfies the original tolerances as well as the *trigger* tolerances which are harder to violate, so perturbational stability is achieved.

4.2.3 Controlled Cumulative Graph Damage (CCGD)

When graph massage is called it makes changes to the graph and thus 'damages' it, in some sense to be defined. This damage must be acceptable for each node annihilation and the *cumulative* damage that is done to the graph must not be unacceptable. The control of this damage is known as *Controlled Cumulative Graph Damage*. An example of a graph that suffers acceptable damage for each nodal annihilation, but suffers an unacceptable amount of cumulative damage can be seen in Figure 4.1.

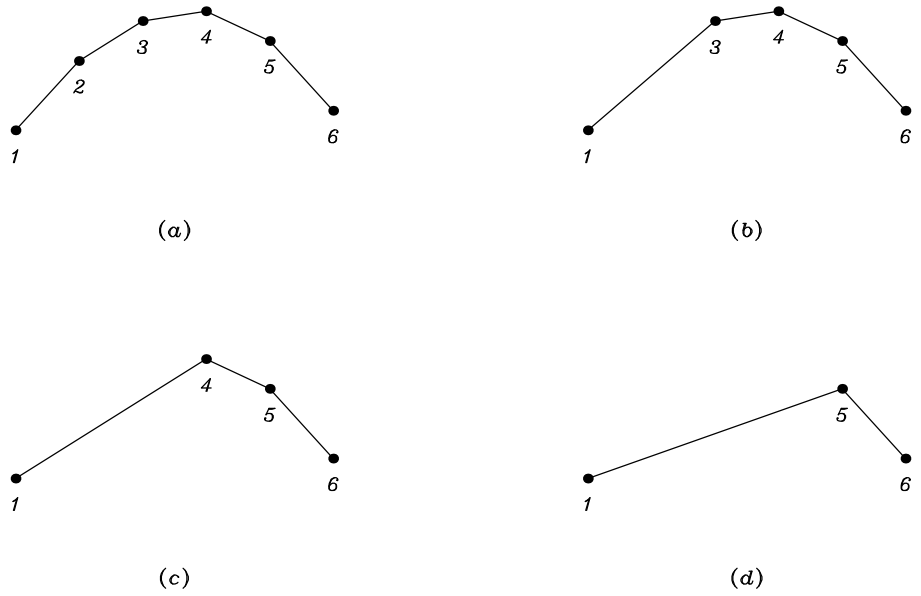

Figure 4.1 Unacceptable cumulative graph damage.

To prevent this kind of unacceptable damage being done, two bounds on the cumulative graph damage have to be satisfied and these are

1) Each point on the original graph G is mapped onto a corresponding point on the new graph G' so that the displacement suffered by this mapping is bounded.

2) The angles between the normals to adjacent faces of the graph G chang by a bounded amount when mapping to the new graph G'.

This means that the points on G are only allowed to move a small distance and that the shape of the graph is only allowed to be distorted by a small amount.

4.3 The Three Graph Massage Modules

Before commencing with the details of the modules it is necessary to describe how the nodes are labelled, how the algorithm keeps track of the nodes and how the nodes are related to each other.

4.3.1 Nomenclature

The left boundary node is labelled '0' and the right bounday node is labelled 'n'. The value of n may vary as the solution evolves, but the left boundary node **cannot** be annihilated as this would be considered to be an unacceptable distortion of the graph. A general interior node is labelled 'i'.

The segment joining nodes $i-1$ and i is labelled $Segment_i$ and $Chord_i$ is the line joining node $i-1$ to node $i+1$. The angle between $Segment_i$ and the $Chord_i$ is $\hat{P}_i$ and the angle between $Segment_{i+1}$ and $Chord_i$ is $\hat{S}_i$.

As nodes are created and annihilated, the FORTRAN 90 code renumbers the nodes so that the nodes are always numbered from 0 to n, and so that node i always lies between nodes $i-1$ and $i+1$. In this way the solution graph is completely described by the positions of the nodes $0, \ldots, n$.

4.3.2 Creation on Length Module

If a given segment, $Segment_i$, is longer than the user-specified maximum segment length, *Max_Segment_Length*, then this module attempts to insert a node. This node will be inserted so as to bisect $Segment_i$, as in Figure 4.2.

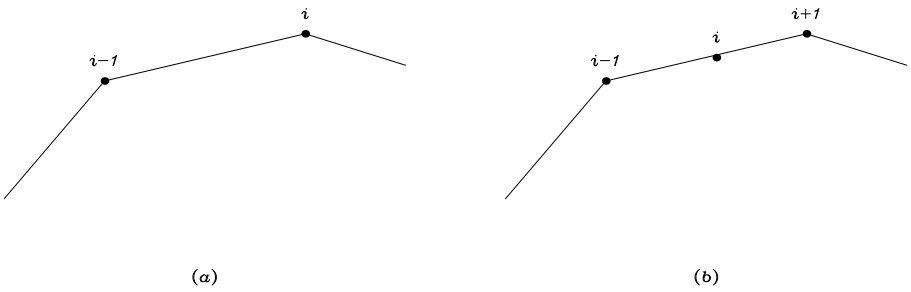

Figure 4.2 Insertion of node by Creation on Length.

The insertion of a node is not guaranteed though, and the failure to add a new node may be owing to one of the following reasons:

1) The maximum allowable number of nodes is already being used.

2) One of the new cell widths, Δs_i or Δs_{i+1}, violates the user-specified minimum cell width tolerance, *Min_Cell_Width*, where *Min_Cell_Width* is used to prevent nodes from being added too close together. (This is useful for preventing the insertion of nodes on shocks or steep fronts.)

The longest segments are bisected first (as they are seen to need it the most), so a rank list needs to be created which contains all of the segments whose length

exceeds *Max_Segment_Length*. This list is then ordered[1] smallest to largest, and segments are taken from the end of the list. If a new node is added then the new segment lengths are checked to see if they exceed *Max_Segment_Length*, and if they do then they are added to the list.

4.3.3 Annihilation on Break Module

Before going any further it is necessary to define what is meant by the term *break*. *Break* is a measure of the displacement damage done to the graph G by inserting or removing a node, and using the definition as given in [7], the break at i, $Break_i$, is defined to be the perpendicular distance between node i and $Chord_i$.

If $Break_i$ is less than the user-specified annihilation on break tolerance, *Annihilation_Brk_Tol*, then the node is added to a rank list of nodes which are marked for deletion. The list is ordered largest to smallest, with each node being taken from the end of the list so that the nodes with the smallest break will be removed first.

As for the creation on length module, the occurence of node i in the rank list does not necessarily mean that node i will be annihilated, and because the annihilation module is "intelligent" there are more criteria which can prevent the annihilation of the node from taking place. The criteria that would prevent annihilation are:

1) The minimum number of nodes is already being used.

2) The incremental angle damage done to G exceeds the user-specified

[1]Every time that nodes are added to the list it needs to be sorted. This should not cause any problems though, as it is expected that relatively few nodes will be created or annihilated at any given time.

tolerance, *Annihilation_Ang_Tol*.

3) The resulting chord segment exceeds the user-specified maximum segment length, *Max_Segment_Length*.

4) The resulting breaks at the adjacent nodes exceed the user-specified creation on break tolerance, *Creation_Brk_Length*.

5) The cumulative break damage exceeds the user-specified cumulative break tolerance, *Cum_Brk_Tol*.

6) The cumulative angle damage exceeds the user-specified cumulative angle tolerance, *Cum_Ang_Tol*.

Node i is removed from the list whether it's deleted or not. If, however, node i is deleted then the break at the two adjacent nodes $i-1$ and $i+1$ needs to be calculated and if either break is less than *Annihilation_Brk_Tol* then the appropriate node needs to be added to the list. The effect of a node annihilation on the graph can be seen in Figure 4.3.

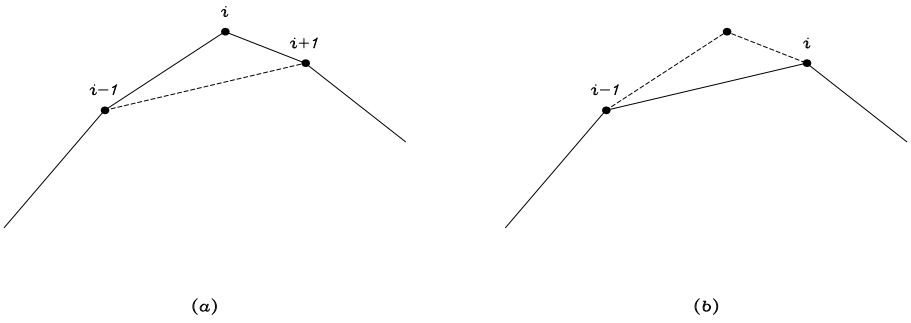

Figure 4.3 Node deletion by Annihilation on Break.

While the annihilation list contains the nodes, and the damage that the nodes would do to the graph if removed, it does not in any way measure the *cumulative* damage that the graph would suffer as a result of removing a node, taking into account any previous annihilations around that node. This *cumulative* damage needs to be measured in some other way and the details of the analysis can be found in [7].

If node i is annihilated then the cumulative break damage and the cumulative angle damage measures are given by :

$$Cum\_Brk\_Ttl_{i-1} = \text{MAX}\,\{Cum\_Brk\_Ttl_{i-1}\,,\, Cum\_Brk\_Ttl_i\} + Break_i$$

and

$$Cum\_Ang\_Ttl_{i-1} = \text{MAX}\,\{Cum\_Ang\_Ttl_{i-1} + \hat{P}_i\,,\, Cum\_Ang\_Ttl_i + \hat{S}_i\}$$

respectively.

4.3.4 Creation on Break Module

Previously, $Break_i$ was perceived as a measure of how expendible the node i was, ie for values of $Break_i$ greater than *Annihilation_Brk_Tol*, annihilation of node i would be out of the question. Here it is used to determine whether or not a node needs help. For values of $Break_i$ above the user-specified creation on break tolerance, *Creation_Brk_Tol*, it is said that node i "carries too much displacement" and that a new node needs to be created next to i to reduce $Break_i$ to a *target* break value. The new node is inserted to the left or right of i depending on which segment is longest, as can be seen in Figure 4.4, and this is done to reduce the difference between segment lengths.

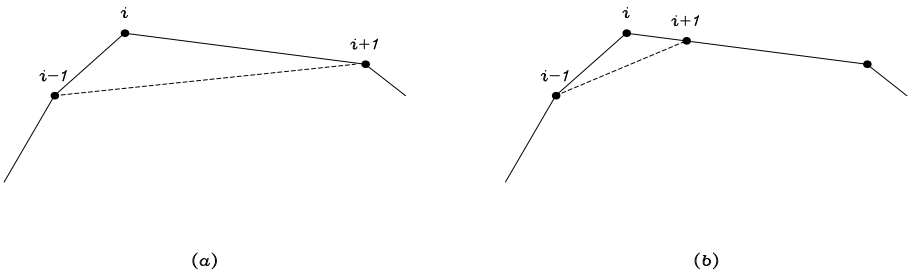

Figure 4.4 Node insertion by Creation on Break.

When calculating $Break_i$ it is important that nodes do not overtake, as this can lead to $Break_i$ becoming a gross underestimate of how much node i "needs help". However, this situation should never occur for a properly formulated PDE which is solved using the GWMFE method with graph massage.

If i is an interior node, then to find the length of the new segment that will reduce $Break_i$ to the *target* break value, Kuprat finds the root of a quadratic equation via an algorithm that "obtains the correct root in every case". However, no details of this algorithm were given, and when looking at the problem of calculating the new segment length, it was discovered that the new length could be found very simply by using the geometry of the triangles involved. The problem reduces to one of calculating two angles and these are easily found by using the known segment lengths, $Segment_i$ and $Segment_{i+1}$, and the known angles $\hat{P}_i$ and $\hat{S}_i$.

If i is one of the boundary nodes, then calculating the new segment length is even easier. For Neumann boundary conditions $Break_i$ is defined to be the

vertical distance between i and the adjacent interior point, so finding the new segment length that will reduce $Break_i$ to the *target* break value is just a matter of comparing the ratio of the break values to that of the segment lengths.

Chapter 5

Results

5.1 Note on FORTRAN 90

One useful feature of FORTRAN 90 that was utilised by the code was that of derived data types, which made it easy to keep track of the nodes that were associated with particular segment lengths, break values and angles in the creation/annihilation routines. In 2-D, where the data structure is more complicated, the use of derived data types would be particularly useful. The use of FORTRAN 90 also made the general coding process a lot easier.

5.2 The Results

In this chapter the graphs from the various FORTRAN 90 programs are presented along with the tolerances that were used to produce each graph. In all of the graphs $\Delta t_{max} = 60$s and the final time is 43200s.

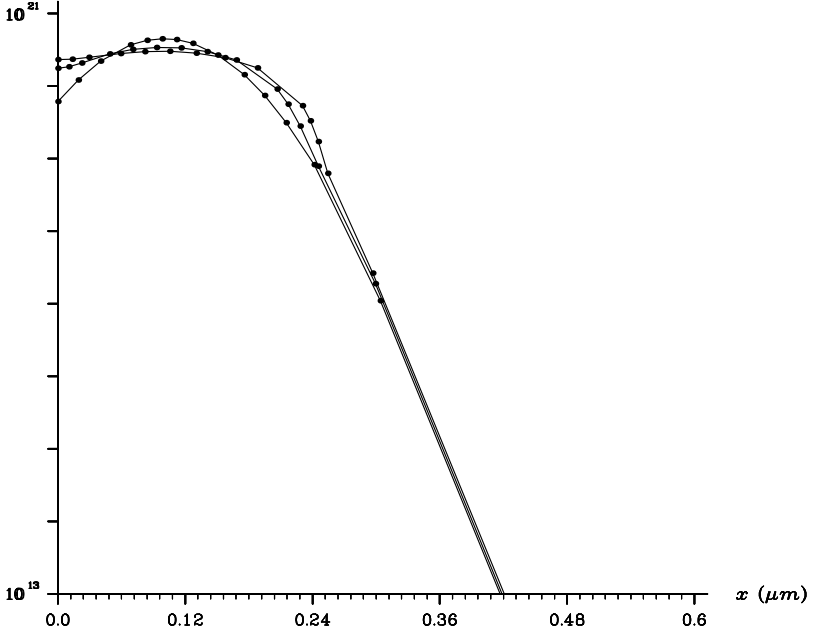

Figure 5.1 MFE with 21 nodes.

The graph above shows the results using the normal MFE method at $t = 0$, 21600 and 43200s.

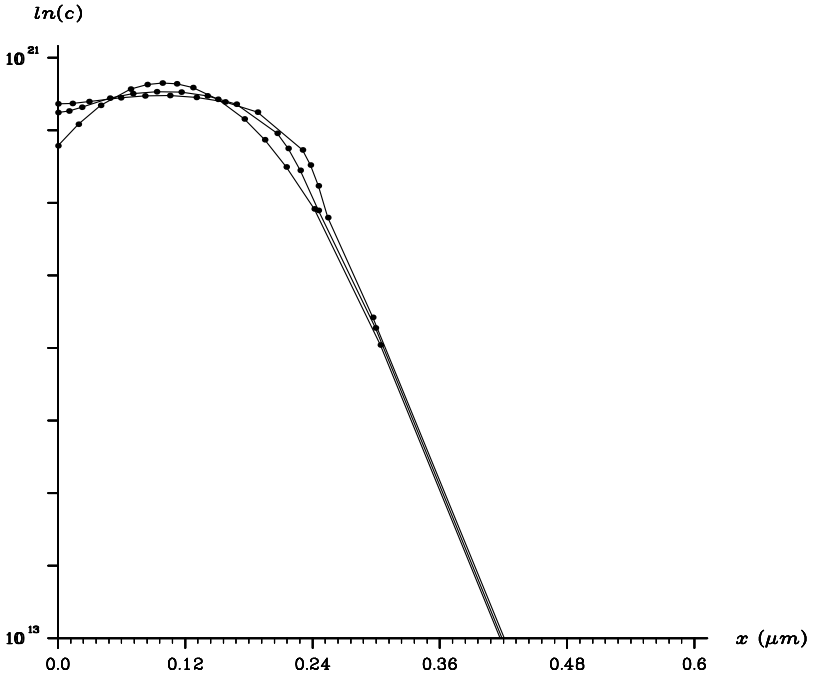

Figure 5.2 GWMFE with 21 nodes.

The graph above shows the results using the Gradient Weighted MFE method at $t = 0$, 21600 and 43200s.

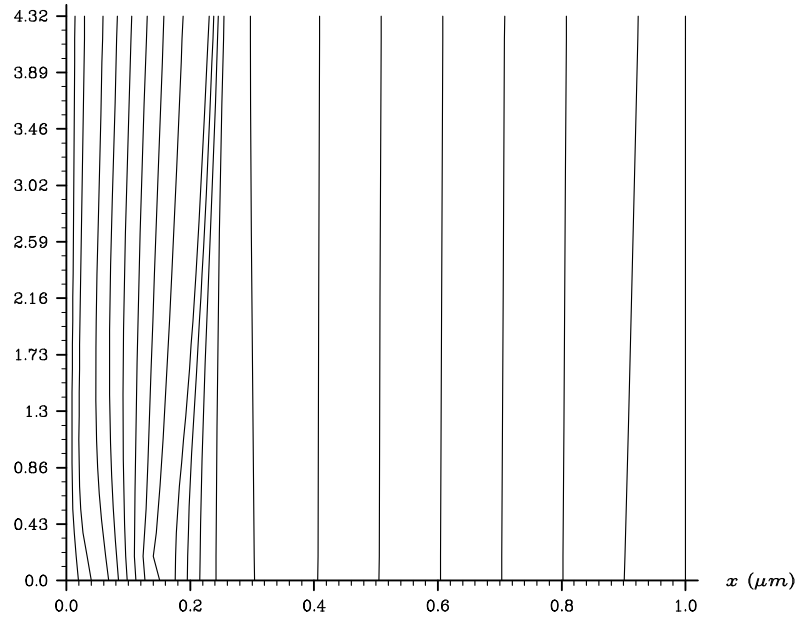

Figure 5.3 MFE nodal movement.

The graph above shows the positions of the 21 nodes up to $t = 43200$s for the normal MFE method.

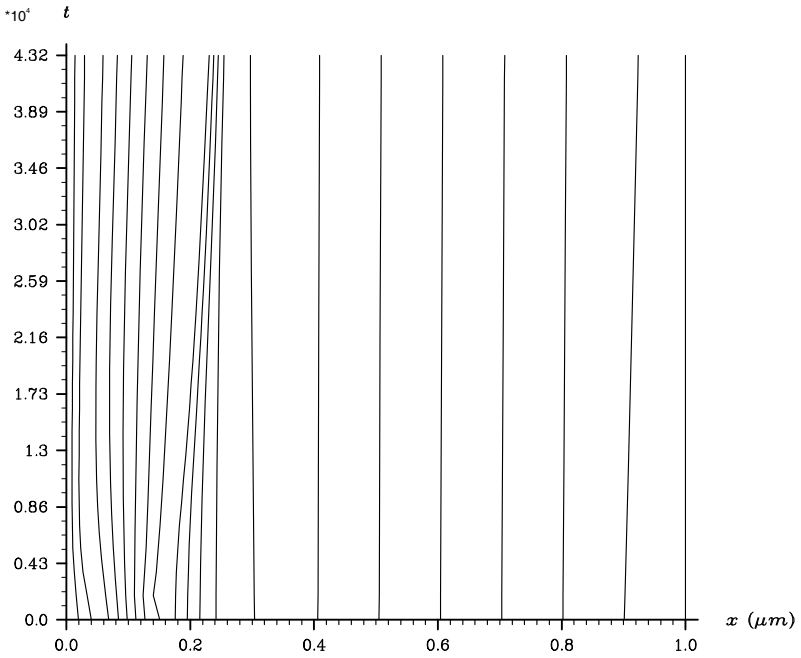

Figure 5.4 GWMFE nodal movement.

The graph above shows the positions of the 21 nodes up to $t = 43200$s for the Gradient Weighted MFE method.

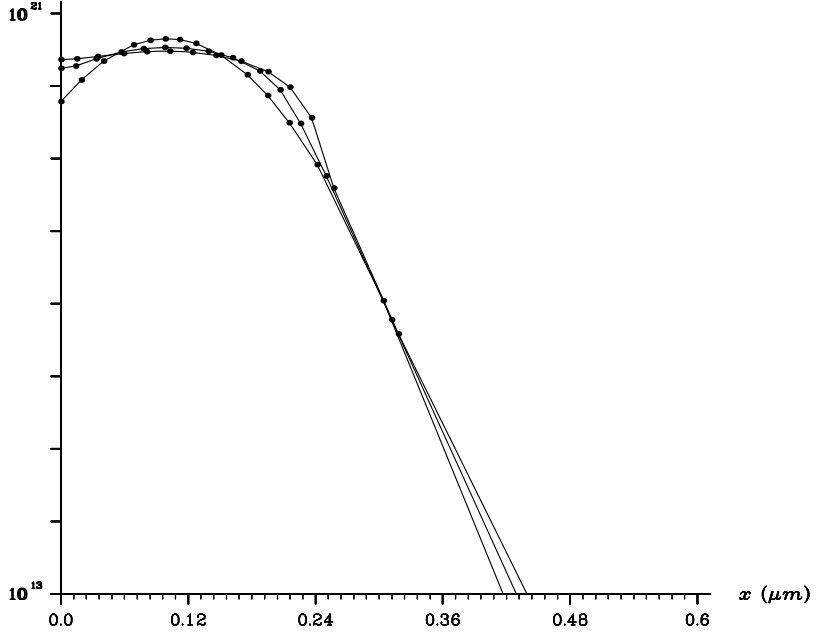

Figure 5.5 MFE with penalty functions.

The graph above shows the solution obtained using penalty functions, where the parameter $\delta = 0.001$.

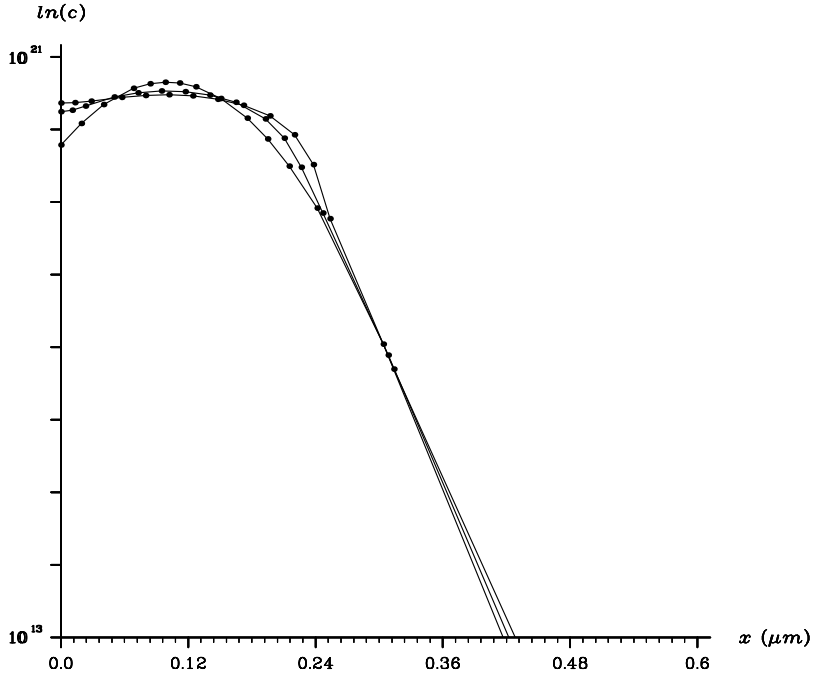

Figure 5.6 GWMFE with penalty functions.

The graph above shows the solution obtained using penalty functions, where the parameter $\delta = 0.0001$.

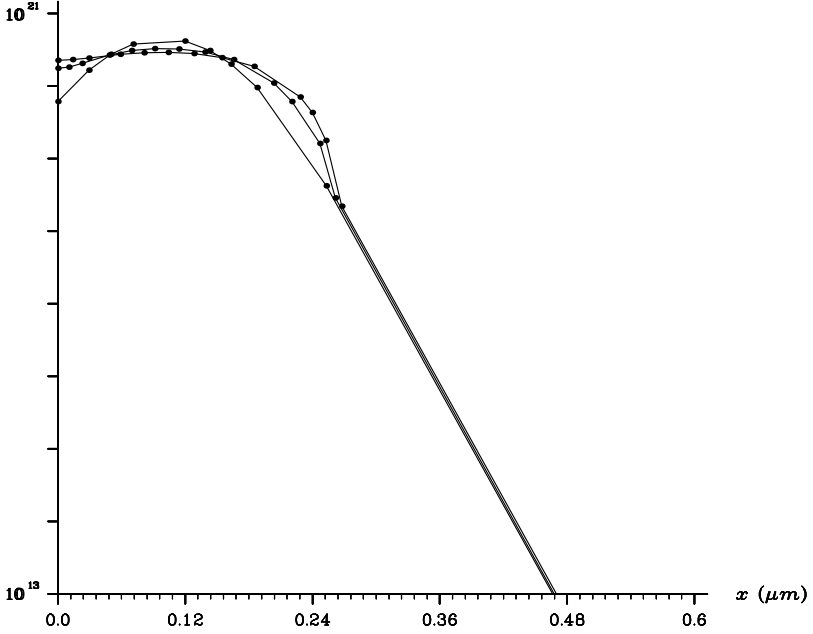

Figure 5.7

The graph above shows the solution obtained using graph massage with the following tolerances :

Min_Num_Nodes	=	15
GM_Frequency	=	5
Trunc_Err_Tol	=	0.001
Creation_Brk_Tol	=	0.05
Max_Segment_Length	=	0.25
Stability_Factor	=	1.7

The graph was massaged 2 times.

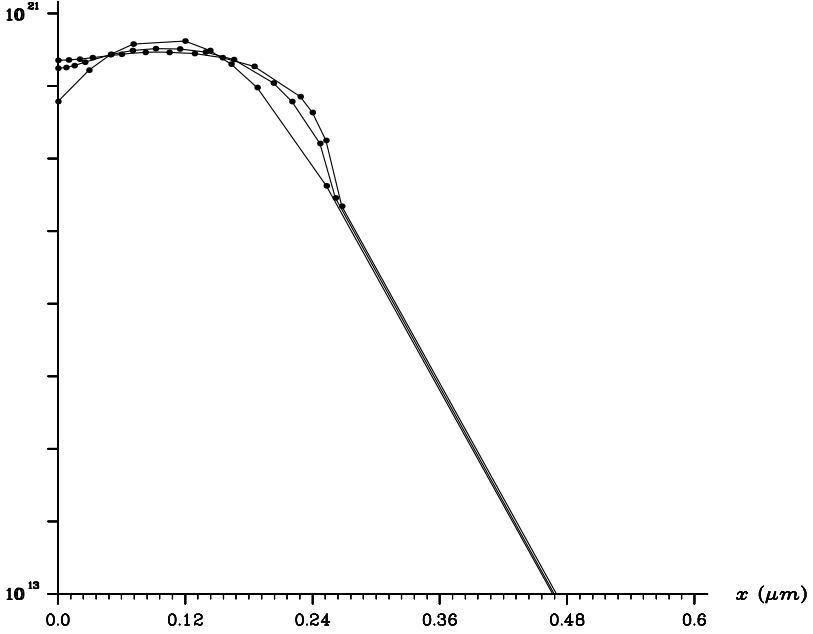

Figure 5.8

The graph above shows the solution obtained using graph massage with the following tolerances :

Min_Num_Nodes	=	15
GM_Frequency	=	5
Trunc_Err_Tol	=	0.0001
Creation_Brk_Tol	=	0.05
Max_Segment_Length	=	0.25
Stability_Factor	=	1.1

The graph was massaged 4 times.

Chapter 6

Conclusions

In this dissertation the method of moving finite elements has been examined, and then applied, to an existing problem in semiconductor process modelling. A program had already been written for the problem several years ago, by P.K. Sweby, but it had been observed that the program wasn't very *robust* in the sense that for more "difficult" cases (ie a different choice of parameters) the program failed to resolve the problem accurately.

The original program was re-written in FORTRAN 90 and then various modifications were made to it in an attempt to improve the robustness of the method. The original problem was retained in order to assess any improvements that might result.

The first modification was the use of weight functions which give rise to the GWMFE method. It was hoped that the gradient weighting would improve the nodal distribution by moving nodes away from the steep front and into the tail region where the resolution is of great importance. However, when comparing Figures 5.1 and 5.2, which are the normal and gradient weighted

outputs respectively, it is not obvious that there is any difference between the two solutions.

Comparison of the nodal positons over time, Figures 5.3 and 5.4, also suggests that GWMFE has not performed any better than normal MFE as the nodal movement is almost identical. The fact there was no nodal movement away from the shock, when using GWMFE, was rather disappointing, and it may have something to do with employing a Neumann boundary condition at $x = 0$. This BC creates a sharp *corner*, where the left shock is 'reflected', and the nodes may be staying in this region in order to resolve this difficulty. Increasing the number of nodes in the approximation had little affect on the approximations.

The second modification was to include penalty functions and this introduced a user-specified parameter δ, which, in some sense, lost a certain amount of the robustness that was being sought. It was necessary to find a *'good'* value of δ that managed to improve the approximation, but it was observed that changing delta caused the position of the tail to change quite drastically, Figures 5.5 and 5.6. This was quite worrying, and the fact that the tail position depended strongly on the choice of δ would seem to suggest that this was not a good modification to make for this problem[1].

The final modification that was made was to use *Graph Massage* which is a new technique that is used in conjunction with the GWMFE method to 'tweak' the graph every few time-steps, in order to stop things from going wrong. The method itself seems fairly straightforward, but unfortunately the method cannot be seen to be robust as it requires a large number of user-specified parameters,

[1]Since the solution did not become parallel when run using the normal MFE method, it was not expected that penalty functions would greatly improve the solution.

and although several of these can be related to each other, at least *four* of the parameters have to be chosen by trial and error. It took several hours of experimention before any parameters were found to work for the semiconductor problem, and even then the results were rather disappointing.

Two plots using graph massage can be seen in Figures 5.7 and 5.8. The first appears to give quite a good resolution of the shock, but there is something strange happening around the tail region. The second appears to give a poorer approximation to the shock, but the 'kink' in the tail region is less pronounced. No amount of fiddling with parameters managed to produce a solution that looked much different from the two shown here.

There is also a "problem" associated with graph massage and this is the fact that the height of the solution should be of the same order as the length of the solution interval. If this is not the case, and the solution height is a lot larger than the the interval length, then the 'dumb' creation on length module will require either a large value for *Max_Segment_Length*, in which case nodes will tend to be placed on any peak in the solution and nowhere else, or the number of nodes will have to be large in order to make *Max_Segment_Length* small enough to resolve details away from any peak in the solution. The result is that the it may be necessary to scale the problem in order to use *Graph Massage* and again this is not a good thing for robustness.

Although the results obtained have not been very promising, the use of graph massage in 2-D would seem to offer the possibility of solving ('fixing') problems, which until now, may have had little chance of being solved. If more time had been available, an investigation of graph massage in 2-D would have been undertaken,

and it is in 2-D that Graph Massage appears to have most applicability.

Bibliography

[1] M.J. Baines, N.R.C. Birkett & P.K. Sweby (1990), *'Non-Linear Diffusion In Process Modelling'*, **I. Jour. Num. Model.**, Vol. 3, pp. 79 - 90.

[2] P. Concus, G.H. Golub, D.P. O'Leary (1975), *'A Generalised Conjugate Gradient Method for the Numerical Solution of Elliptic PDEs'*, in **Sparse Matrix Computations** (Eds. J.R. Bunch, D.J. Rose) Academic Press, New York, pp. 309 - 332.

[3] R.B. Fair (1981), *'Concentration Profiles of Diffused Dopant in Silicon'*, in **Impurity Doping Processes in Silicon** (Ed. F.F.Y. Wang) North Holland, New York, pp. 315 -442.

[4] A.H. Glasser (1988), *'A Moving Finite Element Model of the High Density Z-Pinch'*, **Jour. Comp. Phys.**, Vol. 85 N°1, pp. 159 - .

[5] I.W. Johnson, A.J. Wathen & M.J.Baines (1988), *'Moving Finite Element Methods for Evolutionary Problems. II. Applications'*, **Jour. Comp. Phys.**, Vol. 79 N°2, pp. 270 - 297.

[6] J.R. King & C.P. Please (1986), *'Diffusion of Dopant in Crystalline Silicon: An Asymptotic Analysis'*, **IMA J. Appl. Math.**, 37, pp. 185 - 197.

[7] A.P. Kuprat (1992), *'Creation and Annihilation of Nodes for the Moving Finite Element Method'*, **PhD Thesis**, University of California.

[8] K. Miller (1981), *'Moving Finite Elements I'*, **SIAM J. Numer. Anal.**, 18, pp. 1019 - 1032.

[9] R.O. Moody (1988), *'The Numerical Solution of Moving Boundary Problems Using MFE Methods'*, **PhD Thesis**, University of Reading.

[10] C.P. Please & P.K. Sweby (1986), *'A Transformation to Assist Numerical Solution of Diffusion Equations'*, **Numerical Analysis Report 5/86**, University of Reading.