INTERSTITIAL BRACHYTHERAPY CANCER TREATMENT OPTIMIZATION USING SIMULATED ANNEALING AND ARTIFICIAL NEURAL NETWORKS

By

Steven R. G. Miller

A Thesis Submitted to
The Faculty of Graduate Studies
In Partial Fulfilment of the Requirements
For the Degree of

MASTER OF SCIENCE

Department of Electrical and Computer Engineering
University of Manitoba
Winnipeg, Manitoba, Canada

Thesis Advisor: Professor W. Kinsner, Ph.D., P.Eng.

0-612-79986-7

INTERSTITIAL BRACHYTHERAPY CANCER TREATMENT OPTIMIZATION USING SIMULATED ANNEALING AND ARTIFICIAL NEURAL NETWORKS

BY

STEVEN R.G. MILLER

A Thesis/Practicum submitted to the Faculty of Graduate Studies of The University

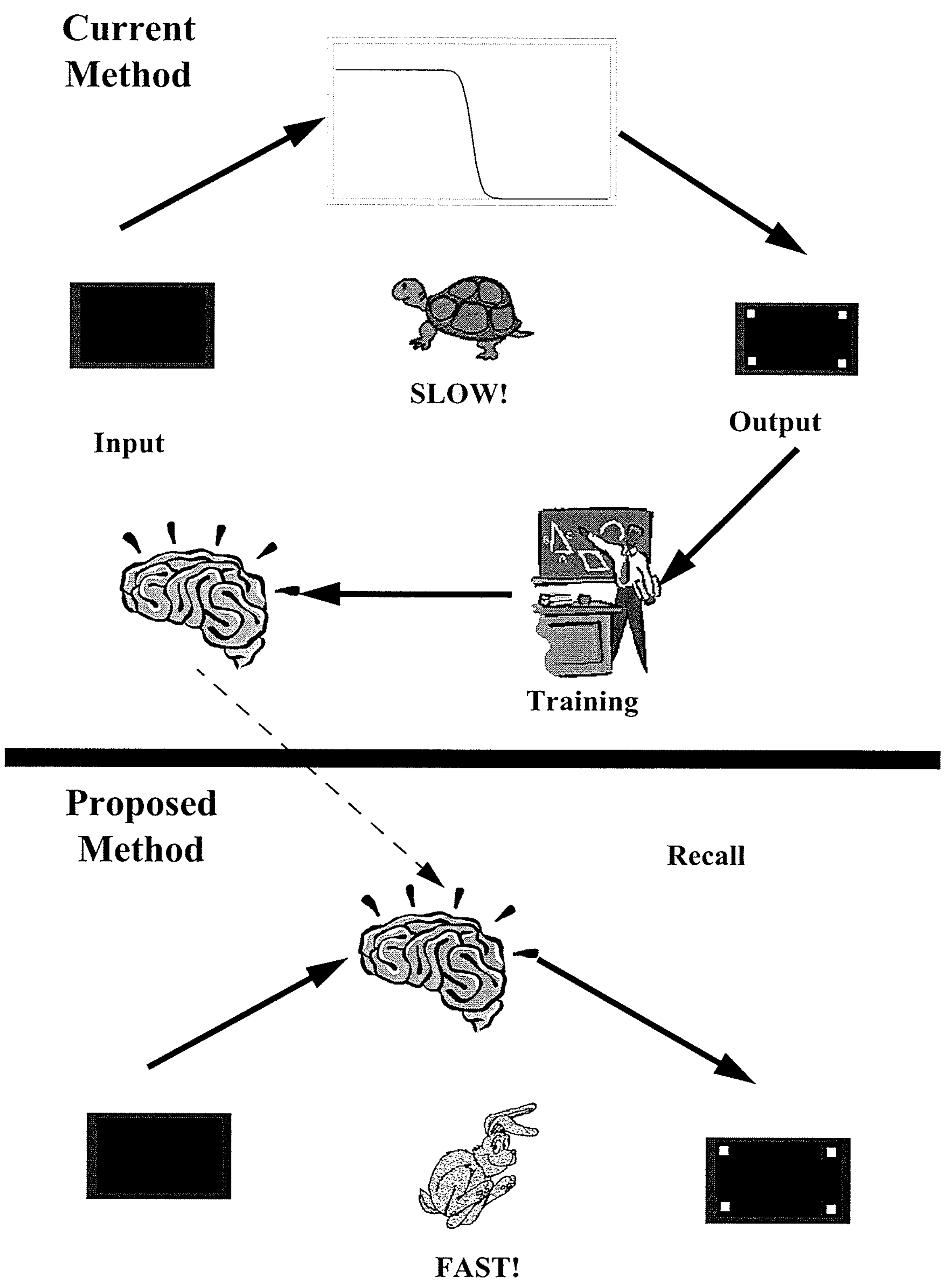
of Manitoba in partial fulfillment of the requirements of the degree

of

Master of Science

STEVEN R.G. MILLER © 2002

This page intentionally left blank.


Current Method
SLOW!
Input
Output
Training
Proposed Method
Recall
FAST!

INTERSTITIAL BRACHYTHERAPY CANCER TREATMENT OPTIMIZATION USING SIMULATED ANNEALING AND ARTIFICIAL NEURAL NETWORKS

By

Steven R. G. Miller

A Thesis Submitted to
The Faculty of Graduate Studies
In Partial Fulfilment of the Requirements
For the Degree of

MASTER OF SCIENCE

Department of Electrical and Computer Engineering
University of Manitoba
Winnipeg, Manitoba, Canada

Thesis Advisor: Professor W. Kinsner, Ph.D., P.Eng.

(xviii + 170 + A23 + B6 + C45 + D34 + E222) = 518 pp.

ABSTRACT

Optimization of interstitial brachytherapy implants has recently turned to non-deterministic optimization techniques, such as simulated annealing (SA) and genetic algorithms (GA). However, the current SA and GA approaches have three major limitations: (i) they are computationally expensive, with the fastest being reported at 3 minutes of dedicated CPU time for a single solution, (ii) they are limited to evaluating seed positions at predefined needle positions, and (iii) they can not be used to update plans during needle insertion. In order to address these shortcomings, a system has been designed and implemented which uses SA and an artificial neural network (ANN). The role of the SA is to find optimal source placements within a tumour from which the ANN can be trained. If the training of the ANN is carried out properly, it is able to generalize the training data, and is capable of computing optimized brachytherapy cancer treatments in milliseconds.

The system developed in this thesis is the first step towards an ANN-based optimization technique for interstitial brachytherapy. The SA is designed to optimize source placement within 2D tumour shapes and produces results that meet the requirements identified by a suitable cost function. The ANN component is designed to generate relative-dose distributions for 2D square tumour shapes using constant source strengths. Through experimentation, it has been determined that the most appropriate structure for the single hidden layer ANN has 12 interior nodes for tumours up to 3 cm in cross sectional size. Using this network layout, the ANN is able to achieve a root mean square (RMS) error of 2.03% of the relative dose on the final pass through the training data, an RMS error of 13.37% on a test set, an average positional error of 1.07 mm and a maximum of 3 mm in positional error, compared to the results created by the SA.

ACKNOWLEDGMENTS

In everyone's life, at some time, our inner fire goes out. It is then burst into flame by an encounter with another human being. We should all be thankful for those people who rekindle the inner spirit.
- Albert Schweitzer (1875-1965)
- French theologian, musician, medical missionary

There are a number of people who have helped to make this thesis a reality. Without all of them this thesis would not have come to fruition. First of all, I must thank Sarah Miller, my wife, who is always there to listen to difficulties I am having, offer friendly advice and for proof reading this thesis a number of times. My mentor, Dr. Kinsner for his guidance and demonstration of the true nature of engineering. Without the direction of Dr. Bews I would never have learned of brachytherapy or come to be challenged by its shortcomings. I must also thank both departments that I work for, Electrical and Computer Engineering at the University of Manitoba and Medical Physics at CancerCare Manitoba, for their services and financial support. All of my fellow graduate students in both departments, with whom I had the honour of studying with and who have helped me out in one way or another. Richard Lee in particular was paramount in guiding me with my studies.

I must also acknowledge my family and closest friends who are always there for me. My Mom and Dad for all of the opportunities they have given me. First and foremost the opportunity to learn, and also for the countless hours of effort they put in to help me.

Thank you all and congratulations - *we* did it!

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

2D	Two (2) Dimensional
3D	Three (3) Dimensional
ANN	Artificial Neural Networks
BP	Backpropagation
BOWANN	Brachytherapy Optimization With Artificial Neural Networks
CCM	CancerCare Manitoba
CP	Counter-Propogation
CPU	Central Processing Unit
DVH	Dose Volume Histogram
GA	Genetic Algorithm
GO	Geometric Optimization
GUI	Graphical User Interface
HDR	High Dose Rate
LDR	Low Dose Rate
LSO	Least Squares Optimization
MANN	Malleable Artificial Neural Network
MB	Mega Bytes
MHz	Mega Hertz
MIPS	Millions of Instructions Per Second

MRI	**M**agnetic **R**esonance **I**maging
OR	**O**perating **R**oom
OO	**O**bject **O**riented
OS	**O**perating **S**ystem
PNN	**P**robabilistic **N**eural **N**etwork
RAM	**R**andom **A**ccess **M**emory
RBF	**R**adial **B**asis **F**unction
RNG	**R**andom **N**umber **G**enerator
SA	**S**imulated **A**nnealing
SAB	**S**imulated **A**nnealing for **B**rachytherapy
SGI	**S**ilicon **G**raphics **I**ncorporated
US	**U**ltra**s**ound
UM	**U**niversity of **M**anitoba
UI	**U**ser **I**nterface
VR	**V**irtual **R**eality

LIST OF SYMBOLS

$\rho(x)$	The probability of x occuring.
γ	The net input to a neuron.
$F(x)$	A generic activation function.
$Y = \zeta(\gamma)$	The sigmoid activation function with γ input.
α_i	Input to a neuron.
w_i	Connection weight for an inter neural connection.
η	Learning rate of an ANN in training
δ	Error in output from a neuron.
X	Input *vector* to a neural network.
Y	Output *vector* from a neural network.
T	Desired or target *vector* for a neural network.

CHAPTER I

INTRODUCTION

"Humanity needs practical men, who get the most out of their work, and, without forgetting the general good, safeguard their own interests. But humanity also needs dreamers, for whom the disinterested development of an enterprise is so captivating that it becomes impossible for them to devote their care to their own material profit. A well-organized society should assure to such workers the efficient means of accomplishing their task, in a life freed from material care and freely consecrated to ***research****."*

- Marie Curie (1867-1934)
- Physicist, Chemist, Discoverer of Radium, 2 Nobel Prizes

1.1 Motivation

The motivation for this research is to improve the quality of life for cancer patients treated with brachytherapy. The use of brachytherapy to treat cancer patients is on the increase, primarily due to advances in technology as well as the ability to treat relatively new treatment sites, such as the prostate gland. Optimizing a brachytherapy cancer treatment increases the probability of killing the cancerous cells and decreases the harmful side effects of radiation for the patient, thus improving the quality of life.

The rate of cancer is increasing and the risks of developing cancer at some point in one's life is approximately 41% for men and 38% for women [NCIC02]. In Canada, approximately 129,300 new cases of cancer and 63,400 deaths were expected to occur in 2002 [NCIC02]. With numbers like these, cancer may soon overtake heart disease as the number one cause of death in Canada. Currently only 3% of Manitoba's cancer patients are treated with brachytherapy [JeKo00], but that number will climb drastically when the large number of eligible prostate brachytherapy patients begin treatment using the newly developed program at CancerCare Manitoba.

It is necessary to continue research aimed at the improvement of cancer treatment methods until we achieve a perfect record of cancer control. Until the day comes that humankind finds a cure for cancer, it is necessary to continue to treat the disease to the highest abilities using the most advanced methods available. The greatest chance of killing the disease comes from treatments that deliver large amounts of radiation to the site of the disease and as little as possible to the surrounding normal tissue. To this end, treatments must be designed specific to each patient [YuSc96]. Developing an automated procedure is the goal of this research project.

1.2 Thesis Objectives and Scope

The goal of this thesis is to identify a procedure for using Artificial Neural Networks (ANNs) in the optimization of interstitial brachytherapy implants. One of the difficulties associated with brachytherapy is that we do not know exactly where to place the radioactive sources in the patient. The placement of the sources is crucial, as it dictates where the radiation is deposited within the patient. Plan optimization involves identifying an arrangement of the sources that deliver a large dose of radiation to the disease and as little as possible to the surrounding normal tissue. If we optimize the placement of the sources, we have a better chance at killing the disease, and sparing the healthy surrounding tissue. The focus of this thesis is on a specific form of brachytherapy referred to as interstitial brachytherapy. In interstitial brachytherapy the sources are placed inside the patient, using needles to create channels through which the sources can travel. The needles are inserted into the patient and through the disease. Since the needles are hollow, the radioactive sources (in the form of "seeds") are slid down the needle into the patient until they rest at the site of the disease. Therefore, the positions of the needles dictate the possible positions for the sources and it is crucial to acknowledge and understand that there is a correspondence between them.

Current methods of optimizing interstitial brachytherapy optimize the placement of the needles, restricting their positioning to predefined template positions [LaBZ00] [PTR96b] [Slob92] [YRPZ98] [YuSc96]. However, this limits ones ability to achieve the optimal treatment plan. Also, as reported by [EJMR98], the needles deviate from the planned positions even when they are inserted into the patient using a template. Ideally then, the positions of the remaining needles should be adjusted in real-time to account for the placement errors of those inserted previously. Currently, there is no research in this area as the standard optimization techniques require far too much time to generate outputs. Using the current techniques, [LaBZ00] [PTR96b] [Slob92] [YRPZ98] [YuSc96], such real-time optimization would require a minimum of 60 extra minutes in the operating room (OR). Therefore, this thesis is to take brachytherapy optimization research in a new direction, using the speed and generalization abilities of ANNs to produce fast optimized source/needle positions within a tumour. The physical templates are replaced with computer graphic representations of them, thereby giving complete freedom to the optimization procedure to place the needles in the most appropriate positions.

This thesis discuses the use of an ANN to optimize the placement of radioactive sources in a tumour. The main research questions include: (i) what to use as input for the ANN, (ii) what to use as output from the ANN, (iii) to ensure that the ANN can actually learn to perform optimization, and (iv) identifying the most appropriate ANN architecture for the optimization problem. Since a three-dimensional (3D) study of thesis issues is too time consuming, this thesis limits the shape of the tumours to squares in two dimensions (2D), but with varying size of up to a maximum of 3 cm per side. Since the tumour representation is 2D, the source positions can be interpreted as the perpendicular applicator positions intersecting that specific 2D plane in the tumour, and therefore the terms source and applicator can be used interchangeably.

Since an ANN must learn from training data, we use simulated annealing (SA) to generate the data. The first step is the generation of optimized source positions within 2D square tumours, using SA. The second step is to decide upon the problem representation for the ANN as well as the structure of the ANN. The final step is to evaluate the effectiveness of the ANN design. The output from the ANN is reported using computer graphics to display the predicted source positions. The sources that the ANN can learn to place are of equal strength, as is typical for permanent implants such as prostate brachytherapy.

1.3 Overview of Chapters

Chapter 2 covers the background information required for the thesis. Topics such as the history of brachytherapy optimization, SA, and ANNs are included.

Chapter 3 covers the software design and discusses the high-level details of the software requirements. The technology mapping and input/output mapping is also addressed.

Chapter 4 deals with the software implementation details, and includes instructions on how to use the developed software.

Chapter 5 discusses the experimental design, and contains an evaluation of the results. Within this chapter, the results of various experiments that guided the design of the software are covered.

The last chapter presents conclusions on the research and makes some recommendations for future work. The limitations of the software are discussed and the contributions that this thesis makes are also highlighted.

CHAPTER II
BACKGROUND

"Not to know what has been transacted in former times is to be always a child. If no use is made of the labors of ***past*** *ages, the world must remain always in the infancy of knowledge."*
- Marcus Tullius Cicero (106-43 BC)
- Roman statesman, orator, philosopher

2.1 Brachytherapy

2.1.1 Definition

The word *brachytherapy* is Greek in origin, and literally translates to *close-distance-therapy*. In brachytherapy, radioactive sources are placed either near, or within a cancerous tumour using various types of source holders called applicators. For example, to place radioactive sources in the lungs, one would feed a long flexible plastic tube into the nasal canal and down into the lungs. The plastic tube then serves as a channel to transport the radioactive sources to the site of the disease. This form of brachytherapy is referred to as *intracavitary brachytherapy*, as it uses a naturally occurring orifice in the body as an insertion channel. In the case of a tumour that does not have a naturally occurring orifice passing near or through it, artificial channels are made by inserting hollow needles through the patient and the tumour. The radioactive sources are positioned either within the needles themselves or the needles are replaced with flexible plastic catheters (which feed through the hollow needles) which then serve as the applicators. This form of brachytherapy is referred to as *interstitial.*

In most instances the radioactive sources are removed after some carefully predetermined period of time. Such a treatment is referred to as a temporary implant. Treatments can also be classified as permanent. In these instances, the applicators are removed leaving the radioactive sources behind in the patient. Clearly, whether an implant is temporary or permanent depends on the characteristics of the radioactive sources and, in particular, the rate at which they deposit radiation in the patient.

Brachytherapy treatments are also categorized as either high dose rate or low dose rate depending on the strength of the radioactive sources. *Low dose rate* (LDR) procedures make use of weak radioactive sources that must remain in the patient for days or months in order to deliver enough radiation to destroy the disease. LDR treatments can be either temporary or permanent. As the source strengths are low, these sources can be handled manually by staff without subjecting them to unacceptable levels or radiation. Treatments which use a very strong radioactive source are referred to as *high dose rate* (HDR) procedures. As the sources are so strong, radiation is deposited in the patient very quickly and treatments last only of the order of minutes. Consequently, multiple sources are not required and a single source can emulate many positions by changing its positions during the treatment. HDR treatments are always temporary and, because of the very strong source utilized they, must be administered via a computer control to ensure that the dose to staff remains below an acceptable level (in remote afterloading, as it is referred, applicators are positioned manually in the patient and then connected to a shielded safe; the source is then transferred via a computer control from the safe to the applicators once the staff have left the treatment room.

2.1.2 History

Brachytherapy is the oldest form of cancer treatment. It began within three years of the discovery of radium by Marie Curie (whose inspirational words are quoted at the beginning of Chapter 1 of this thesis) in 1898. When Pierre Curie (Marie's husband) attributed the erythema on Henri Becquerel's skin to the vile of radium salt he carried in his pocket, he correctly conjectured that it was the biological effects of radiation on tissue. As a result, Pierre suggested that a small tube filled with radium salt be used to treat a patient's tumour [Godd88].

The first interstitial and intracavitary applications occurred between 1905-1915, at a number of institutions in Europe and North America. Initially, glass tubes of radium salt were used, along with flat applicators coated with radium and sealed with varnish. Unfortunately, the clinical experience with these sources revealed that the intense beta (β) radiation emitted simultaneously with the gamma (γ) rays responsible for the treatment of the disease caused tissue necrosis (the localized death of living cells). It was not until 1920 that researchers were able to filter the β-rays successfully by placing radon in small gold tubes. Over time it was realized that a correlation between the biological effects and amount of radiation utilized had to be found. To this end, tables of dose values were created for various combinations of equal strength linear sources. This was the common practice until the 1970s and early 1980s when computers became involved in the process of dose calculations.

Brachytherapy is growing in popularity as it has become less of an art and more of a science due to advances in technology, such as the remote afterloading of HDR brachytherapy sources (as described in Section 2.1.4). There is also an increase in brachytherapy use due to the recent inclusion of once difficult treatment sites, such as the prostate gland. Accurate brachytherapy treatments of the prostate can now be performed as an out-patient procedure using a procedure in which LDR source carrying needles are inserted into the prostate through the perinium under ultrasound or MR guidance. Once the needles are properly positioned, they are removed leaving the seeds behind in the gland. This permanent procedure has proven to be just as successful as radical prostatectomy, with less morbidity for early stage disease.

2.1.3 Typical LDR Procedure

In the LDR brachytherapy, multiple sources of equal strength are placed inside an applicator which serves to hold the sources in a fixed position within the patient. The distribution of radiation within the patient is determined by the geometric arrangement of the sources and, therefore, can be customized for each patient by adjusting the location of the applicators as well as the position of the sources within the applicators.

In a typical procedure, the patient would be imaged prior to the insertion, in order to determine where the applicators should be placed. The act of determining where the radioactive sources should be placed is called treatment planning. The insertion of the applicators would then take place after this initial planning in the OR. As reported in other works [EMJR98], the applicators may deviate from the planned positions to the detriment of the desired distribution of radiation within the patient.

2.1.4 Typical HDR Procedure

In the HDR brachytherapy, a single radioactive source is used to simulate multiple sources. This single source is under computer control and enters the applicators in the patient sequentially. The source is able to be left at a location (referred to as a dwell position) for any length of time (referred to as dwell time). By letting it sit at a specific location for a longer period of time, it emulates a stronger source at that location. Since the distribution of the radiation within the patient is dependent on the combination of dwell position and dwell time, the radiation distribution can be customized for each patient by adjusting those two parameters. In fact, dwell time can be adjusted to compensate for the errors associated with applicator placement to some degree. This additional parameter of dwell time complicates the optimization problem significantly, and thus this thesis focuses on the second form of brachytherapy, LDR because of it relative simplicity.

2.1.5 Advantages of Brachytherapy

Brachytherapy has a number of advantages over the more common external beam treatments (with the source of radiation located external to the patient). The goal of radiation therapy is to deliver as much radiation as possible to the disease and as little as possible to the surrounding healthy tissue. If not, the treatment will be associated with unacceptable morbidity. External beam treatments are fundamentally at odds with this goal as the radiation must first travel through normal tissue to reach the underlying disease. This necessarily results in radiation being deposited in healthy tissue and in fact, due to the nature of radiation interactions, more than the disease itself. On the other hand, brachytherapy places the radiation source in the tumour and, therefore, does not suffer from this shortcoming. In addition, brachytherapy makes use of the inverse square law principle whereby the radiation decreases inversely with the distance squared from the source, as shown in Fig. 2.1.

In external beam therapy the patient is relatively far from the source, and the falloff in radiation with distance travelled through the patient is described by this portion of the curve to the far right of Fig. 2.1. Therefore, the amount of radiation deposited at the entrance of the patient is comparable to the amount of radiation deposited at the site of tumour as well as at the site that it leaves the patient. External beam treatments must deliver radiation with beams (beams entering the patient at different locations) in order to increase the radiation at the tumour relative to that deposited in the healthy tissue. In brachytherapy however, the source is either extremely close to or inside the tumour, and the radiation in the tumour is considerably higher than the radiation being deposited in the surrounding structures, as illustrated be the portion of curve on the far left of Fig. 2.1. Therefore, using brachytherapy there is a better chance of sparing the healthy surrounding tissue and isolating the radiation to the affected area (tumour).

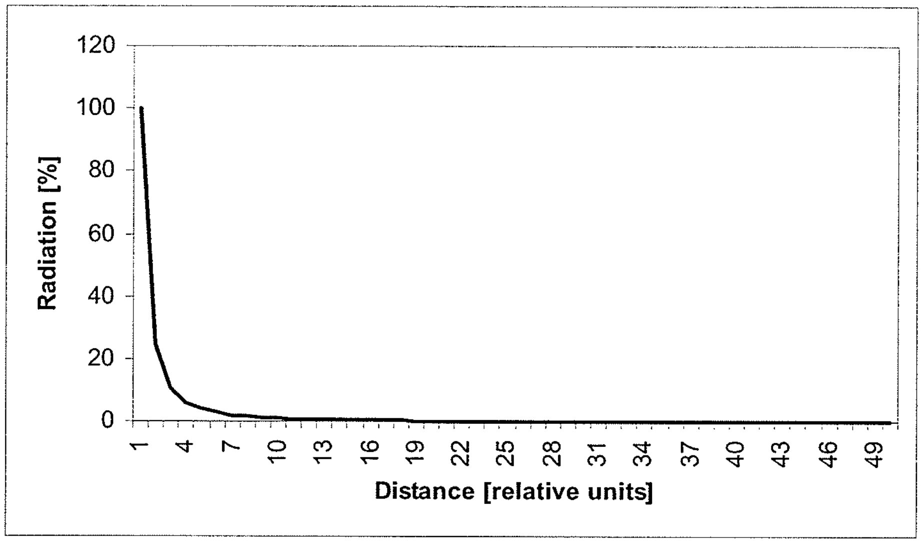


Fig. 2.1 Radiation as a function of distance.

2.1.6 Disadvantages of Brachytherapy

Unfortunately brachytherapy has a number of disadvantages, making the choice of using brachytherapy much more difficult. First of all and most obviously is the fact that brachytherapy is invasive, as applicators are inserted into the patient: either into naturally occurring cavities, or those made by inserting needles through tissues. Secondly, it can be very time consuming due to several factors, including: (i) the time required in the OR to insert the applicators, (ii) the fact that most brachytherapy patients must stay in the hospital during their treatment, and (iii) that it often takes an entire day to plan the location of the applicators and sources once the patient has been imaged. Due to the inverse square law, delivering adequate radiation to all aspects of most diseases sites requires that many sources be positioned throughout the treatment volume. Determining the optimal position of these sources is tedious, especially considering the fact that altering any of the sources by a few millimeters can have a dramatic effect on where the radiation is delivered due to the inverse square law.

2.2 Common Optimization Techniques

This section will summarize various optimization techniques. The reason that they are covered at this specific point in the thesis is that the section following will take a detailed look at the optimization techniques applied to brachytherapy past and present. The specifics of these techniques are not crucial for the reader, but some basic understanding of the theory is required to fully appreciate the review.

The dictionary definition of optimization is "the procedure or procedures used to make a system or design as effective or functional as possible, especially the mathematical techniques involved" [Dict00]. In mathematical problems, optimization is typically the process of finding the best possible solution to the problem. There are two main methods of optimizing mathematical problems, deterministic and non-deterministic. Deterministic solutions "describe a system whose time evolution can be predicted exactly" [Dict00], whereas non-deterministic methods would not be predictable exactly. There are a number of both types of optimization techniques but only the major methods will be described in this review.

2.2.1 Deterministic Optimization Techniques

Examples of deterministic optimization are the steepest descent (gradient descent or least squares optimization, LSO) and the greedy algorithm. In both techniques, the algorithm always proceeds in the direction that improves the solution the most. For example, optimization of the function F(x) of Fig. 2.2, with a starting point A, returns B as the solution. However, the best solution is at C. Thus, deterministic techniques are very susceptible to their starting point and tend to get stuck at a local minimum, rather than finding the global minimum, as illustrated in the above example [Padb99].

2.2.1.1 Gradient Descent

Gradient descent optimization is a technique that uses the gradient (slope) of a function to find the minimum value [Padb99]. If we have the function y = F(x) as shown in Fig. 2.2 and we start at point A, we can calculate the slope around A using dy/dx. Once the slope has been calculated, the optimization algorithm travels in the negative slope direction towards the minimum. The starting value of the gradient descent technique completely dictates the minimum that is found as the algorithm only locates local minima. There is no way to find the global minimum at C of the function without modifications to the algorithm.

2.2.1.2 Greedy Algorithm

The greedy algorithm is an algorithm in which we always make a change that has the best immediate outcome. So if y_1 = f(x+dx) and y_2 = f(x-dx) with $y_1 < y_2$ then the greedy algorithm chooses y_1 as the next solution to the problem. If we start at point A in Fig. 2.2 and call the greedy algorithm recursively, it finds the local minimum B eventually. Again, as with gradient descent, the algorithm has no simple way to find the global minimum at C.

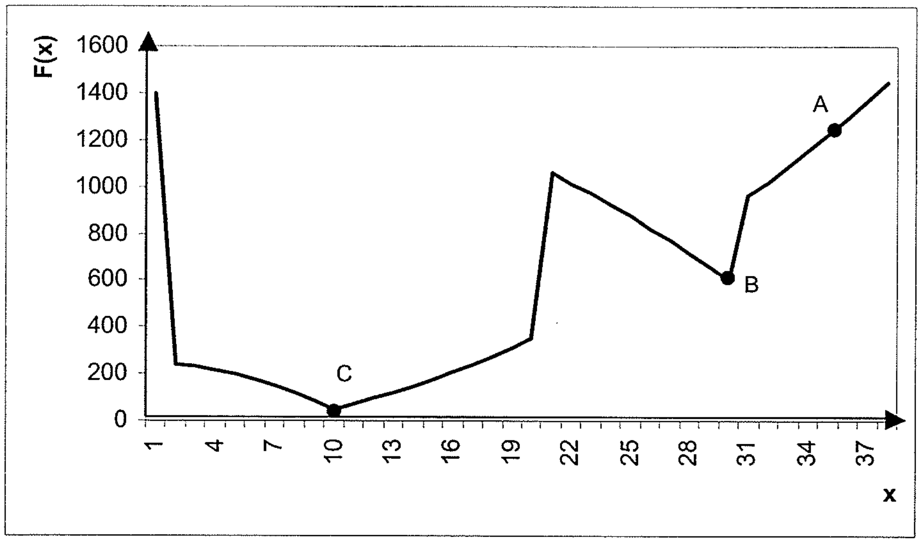


Fig. 2.2 Sample error space for optimization.

2.2.1.3 Least Square Optimization (LSO)

LSO is an optimization technique based on minimizing the squared difference between the result obtained and the ideal case. It is used in situations in which we are trying to approximate a complex function with a more simple equation (typically it is reduced to a line). The LSO is useful when trying to chose between a number of different possible solutions to a problem as it identifies the solution that is closest to the ideal case.

2.2.2 Non-Deterministic Optimization Techniques

Non-deterministic optimization techniques are equipped with schemes of escaping local minima in search of the global minimum. The two most common non-deterministic optimization techniques are the genetic algorithm (GA) and simulated annealing (SA).

2.2.2.1 Genetic Algorithm (GA)

In the GA the problem being solved is mapped into chromosomes, such as those found in DNA. The chromosomes are evaluated using a fitness function and reproduce with other chromosomes based on their fitness. At certain points in the algorithm, the chromosomes may have random mutations that may improve or deteriorate the fitness of the chromosome. The analogy stems from Darwin's theory of evolution and the survival of the fittest. Since the GA is very slow, it is not used in this thesis.

2.2.2.2 Simulated Annealing (SA)

As SA is used for a major portion of this thesis, it will be explained in detail in Section 2.4. Both SA and GA are based on the evaluation of a cost function (fitness function). Therefore, the cost function is an integral component of these non-deterministic optimization techniques. It should be noted that the cost function is problem dependent.

2.2.3 Other Optimization Techniques

There are other classes of optimization techniques, which cannot be clearly classified into deterministic or non-deterministic methods. For example, a specific form of ANN called the Hopfield ANN can also be used to optimize functions, and is covered in Section 2.5.6.

2.3 Past Brachytherapy Optimization Approaches

The literature shows a clear division in optimization techniques, primarily due to technological advances in computing. The inaugural papers of the early 1980s focus on 2D solutions. This is because the computing power necessary to solve the considerably more complex 3D cases (second category) was not available, and because they were new concepts that had to be first proven. The literature makes reference to these two distinct eras as the distance and volume implant methodologies. In a distance implant "dose points" are placed at prescribed positions around the implant area. The computer algorithms then finds a set of dwell positions (and times for HDR) that would yield equal amounts of radiation at those points. No consideration was made for the dose between the dwell positions or the dose to the actual patient anatomy, as the computer systems would have been sufficiently burdened with these few points. In the early 1990s however, the methods of brachytherapy optimization changed to 3D volumes with the goal to achieve a homogeneous dose throughout the tumour, not just at the discrete dose points (as is the case in distance implants). The optimization methods from both approaches will be reviewed, with the primary focus on the newer volume techniques.

Distance brachytherapy optimization is primarily concerned with the dose contribution from M sources to N dose points. The problem stated in this form clearly lends itself to some form of numerical optimization technique. In these techniques every possible position for a source (M) is considered and a dwell time is calculated for each position M. The biggest challenge of these techniques is to find algorithms that do not return negative values for time at some of the positions M. A number of different LSO algorithms have been proposed [RWCA91], [WaAn97], and [BSSL88]. Pistorius and Groenewald [PiGr84] utilized a combination of Gaussian elimination and LSO to find the dwell time for a single stepping source remote afterloading system. Any negative values for time were suppressed to 0, and the algorithm was run again until no negative values were obtained. The results of the study were very good, and this technique was able to obtain a radiation distribution which only differed by only 5% from the prescribed isodose curve (or ideal radiation distribution) within the treatment site. Another technique [VaDe90] utilized singular value decomposition (SVD), and suppressed the large fluctuation in adjacent dwell positions. Again, by running the algorithm a number of times, negative values were eliminated. Unfortunately, these techniques do not lend themselves too readily to the much more difficult problems in 3D and thus different approaches were required.

In the early 1990s new methods began to emerge for volume implants with two techniques: geometric optimization (GO), and simulated annealing (SA) being the early favorites. The GO was introduced in 1990 by Edmundson [Edmu90] [ERTB93] [EdYM95]. In the GO technique, a source's strength is determined by its proximity to other source positions. Clearly, the GO technique is very much like the techniques of the earlier 1980s in that is still relies on deterministic methods of optimization. Although the technique exhibited "unexpectedly good behavior" over the previous optimization methods for 3D implants [AEAA97] [KVDN94], the technique never seemed to flourish as it was based on deterministic optimization techniques.

It was the arrival of the non-deterministic approaches that overtook the 3D volume optimization of brachytherapy by storm. The first report of a non-deterministic approach for brachytherapy was in 1992 by Sloboda [Slob92]. Sloboda used SA to optimize dwell positions for a set of specified dose points. His initial approach was very similar to the approach used in the 1980s in that it was 2D and required the user to input coordinates of dose points. However, his initial paper had a profound effect on the brachytherapy optimization community as it was the first to use SA for volume implants. Since this paper, there have been several others who have published work on using SA and GA for volume implants.

The first SA approach that used patient specific anatomy data in its optimization was introduced by Pouliot *et. al.* [PTR96a] [PTR96b] [PTRV97]. The focus of this work is the optimization of prostate brachytherapy LDR implants. Prior to the actual insertion (usually a couple of weeks), an optimized plan is developed based on the patient anatomy even though patient anatomy changes between the planning and the actual insertion. Pouliot's cost function employed has three factors: (i) a prescribed dose to accomplish; (ii) a uniform dose within the prostate; and (iii) a limited dose to the urethra (which happens to pass through the center of the prostate making for a difficult optimization problem). As it is common practice in prostate brachytherapy to use a template to guide the applicators, Pouliot used the holes in the template as the possible applicator locations. This of course has the added benefit of reducing the extreme complexity of unconstrained brachytherapy optimization. Since the decision by Pouliot, to restrict the applicators to the template positions, all other research has followed suit. Pouliot obtained satisfactory results in his early work achieving optimized implants in 15 minutes on a SUN SPARC 5 workstation. The prostate was the only treatment site to use these advanced optimization techniques until late 1999 when Lahanas *et. al.* [LaBZ99] applied it to the breast and lung.

The first application of the GA to brachytherapy was Yu *et. al.* in 1996 [YuSc96]. In his introductory paper, Yu mapped the brachytherapy problem into a GA solution space, again to generate a pre-plan for the OR. The fitness function that he used is a combination of the dose coverage, the conformity of the dose to the target volume and the number of needles used. The GA that Yu implemented took 30 minutes on average on a SUN SPARC 5 workstation to produce output. When analyzing the results, Yu found a dosimetric improvement in the minimum target dose as compared to non GA optimized treatments, suggesting a better tumour cell kill rate.

Two years after the initial GA paper, a new approach was proposed by Yang *et. al.* [YRPZ98]. They evaluated the effectiveness of utilizing a different representation of the GA to increase the speed of the optimization. Three paradigms were evaluated, the sGA (small GA), the sureGA (small-uniform-restart-elitist GA), and the securGA (small-elitist-creeping-uniform-restart GA). Essentially, these paradigms are ways of avoiding premature convergence on the near-optimal region. They stem from research using very small population sizes in the GA, for problems with a large number of parameters. Using small population sizes and restarting the GA in very few generations (typically 5 - 10), while keeping the overall best solution, produces a faster convergence. Yang's fitness function involves maximizing the minimum peripheral dose, maximizing the uniformity of the dose within the tumour, and minimizing the dose to the critical structures (urethra). The securGA was very effective and was able to produce acceptable results in 5 minutes on an HP735 workstation. However, this approach was still only used to create a pre-plan to be carried out in the OR and was not used iteratively during the insertion.

In the late 1990s, Lahanas *et. al.* published a paper in which they manipulated the GA to improve the performance and introduced the concept of multiobjective GAs to brachytherapy optimization [LaBZ99]. One of the major problems with the fitness and cost functions of the previous techniques is their reliance on user selected scale factors to combine the various terms in the cost function. These factors have a profound effect on the results of the optimization. In multiobjective GA, the algorithm also finds the best scale factors. For more information on multiobjective GA, see the work of Kupinski *et. al.*

[KuAn99]. In the implementation by Lahanas *et. al.* the algorithm returns a number of optimized functions. It returns the absolute best solution based on the Euclidean distance from the goal fitness function, and the solution that best satisfies each of the optimization objectives. In this manner, the oncologist and medical physicist can choose the best solution for the specific case in question. The authors do not report any data for the execution time, which may be quite lengthy considering the nature of the technique. The authors state "thanks to the rapid development of computer hardware this approach will *someday* be a viable approach."

In the late 1999, a paper was published by Messing *et. al.* that combined advanced visualization methods and the GA to optimize prostate implants [MZRB99]. The implant volume is visualized using a combination of virtual reality (VR) and ultrasound (US). The implant is optimized using the GA techniques in Yu's work [YuSc96] and the multiobjective GA techniques previously discussed. The fitness function that is used is a combination of the minimum peripheral dose, uniformity, number of needles, and the maximum dose to critical structures. The optimization of the sources is actually carried out in the OR at the time of the implant. This is the first paper reporting such a result. The average amount of time required by the GA optimization on a trial of 10 patients was 4.2 minutes. The plans optimized in the OR had no dosimetric inferiorities to plans that were optimized ahead of time using more conventional approaches, thus indicating that this approach, and subsequently other methods like it are feasible.

Over the last decade, there have also been some unique approaches to the optimization of brachytherapy implants. These approaches typically use methods borrowed from other areas of study, such as imaging. In 1991, Holmes *et. al.* attempted to optimize implants by applying a deconvolution with a kernel [HMSR91] to the ideal dose distribution. In this manner they were able to obtain a weight distribution corresponding to the total energy per unit mass in the treatment region (activity distribution). Similar problems to those using the LSO arise using this technique, as negative times occur and additional manipulations are necessary. No further work using this technique has appeared in the literature. One final technique using a unique approach is the work of Alfredo *et. al.* in 1997 [ASEF97]. Alfredo used a backprojection algorithm to find the optimal source distribution. However, the results were fairly poor, and since the initial paper, no further work has been attempted using this method.

2.3.1 Critique of Past Brachytherapy Optimization Techniques

Although some of the recent papers on optimization in brachytherapy have presented very good results, they all have one common pitfall: they all assume the use of templates for the insertion of the needles and sources. By constraining the needles and sources to specific predetermined locations, it reduces the complexity of the optimization considerably, and ultimately the time to achieve a solution. The optimization problem becomes much larger with each possible needles or source location. Research results reported in [BOCS90] confirm that customized templates improve the dose distributions within the tumour. Therefore, the ideal procedure must make use of both custom templates (allowing the needles and sources to be placed anywhere) and computer optimization.

2.3.2 Derived Requirements for Brachytherapy Optimization in This Thesis

In reviewing the literature, we can see that the optimization of brachytherapy can have a number of requirements, each represented as a term in a cost function. The requirements for optimization are primarily based on clinical experiences (and common sense to some degree). The main focus is always on the homogeneity of the radiation within the tumour. The next most common requirement in the literature is limiting the amount of dose external to the tumour. The most recent papers also attempt to limit the dose to critical structures in the region of the tumour (such as the urethra in prostate treatments). Some of the papers also acknowledge the need to minimize the number of applicators used in the treatment to avoid extra damage to the treatment area.

Vicini's work [VJHE98] showed that the future of brachytherapy optimization will indeed utilize advanced imaging techniques. We have also demonstrated in other research conducted at CancerCare Manitoba [MBBM98] [MJBK99], that VR and computer graphics have a future in brachytherapy. It is most likely that the future will include customized computer graphics templates that will change in real time dynamically, based on what has already occurred in the insertion of the preceding needles. All of the current optimization techniques require far too much time for real-time applications. For instance, in a typical prostate insertion, there are in the order of 20 needles and the fastest reported time for a prostate optimization is 3 minutes [MZRB99]. Therefore it would require 60 minutes to optimize the treatment after each applicator is inserted in the operating room (OR) (20 needles times 3 minutes each). This is far too much time spent in the OR waiting for computer optimization output. Thus, all of the past techniques fall short, and a new approach must be found. All of these requirements will be included in the process of optimization developed in this thesis.

2.4 Simulated Annealing

The physical act of annealing has been around for centuries, and stems from the shaping of metals into usable forms. For example a blacksmith making a sword would heat solid metal into a liquid state, having a very high internal energy. The liquid metal would then be poured into a cast. In order to give the sword strength, the blacksmith would cool the metal very slowly, allowing the molecules within the metal to align into a crystal structure, the strongest molecular arrangement. It should be mentioned that if the metal is cooled too quickly, quenching occurs (where molecules are trapped in a highly irregular arrangement), and the metal becomes brittle. This process was mapped to a computer optimization technique in the early 1980s by Kirkpatrick *et. al.* [KiGV82] and independently by Cerny [Cern85]. They proposed the following model for optimizing difficult problems using computers [AaKo89].

First, the problem is mapped into a minimization problem (as opposed to a maximization problem). In other words, a single function must be created such that a smaller resultant corresponds to a better solution to the problem. This function is most commonly referred to as the *cost function.* It should be noted that the problem may easily be mapped into a maximization problem, but the classical form of SA involves minimization. The next necessary element is the ability to morph the current solution to the problem into another solution using random decisions. In other words, given the current solution to a problem T_i, we must be able to randomly generate a new solution T_{i+1} which is either be a better solution than T_i or worse. In the case that T_{i+1} is better than Ti, then T_{i+1} becomes the new current solution to the problem. However, in the case that T_{i+1} is worse than T_i, then T_{i+1} is kept with a certain probability, which is based on the current temperature in the simulation. The probability that the new solution is kept is given by

$$p(T_{i+1}) = \exp\left(\frac{(c(T_i) - c(T_{i+1}))}{k_b t}\right) \tag{2.1}$$

where $c(T_i)$ is the cost of the previous solution, $c(T_{i+1})$ is the cost of the current solution, t is the current temperature in the simulation, and k_b is the Boltzman constant, although other constants may be used with the effect of changing the probability distribution function.

A software flowchart of the SA algorithm is shown in Fig. 2.3 and Fig. 2.4. An initial solution to the problem is generated. Next, a new solution is generated based on the initial solution. If the new solution is better than the previous solution, it is kept as the solution to the problem. If the new solution is not better than the previous solution, it is kept with a certain probability based on the current temperature of the simulation. This process is continued until the stopping conditions are met.

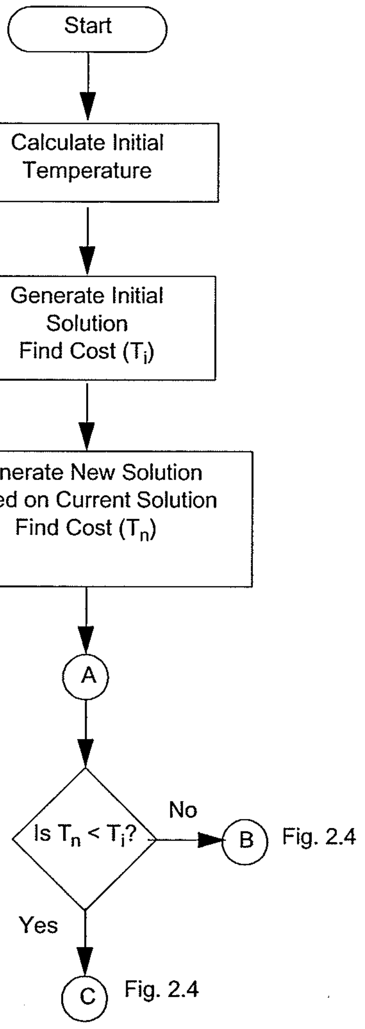

Start
Calculate Initial Temperature
Generate Initial Solution Find Cost (T_i)
Generate New Solution Based on Current Solution Find Cost (T_n)
A
Is $T_n < T_i$?
No
B
Fig. 2.4
Yes
C
Fig. 2.4

Fig. 2.3 Flowchart of the SA algorithm - part 1.

.Since the temperature in Eq. 2.1 is what ultimately determines the probability that

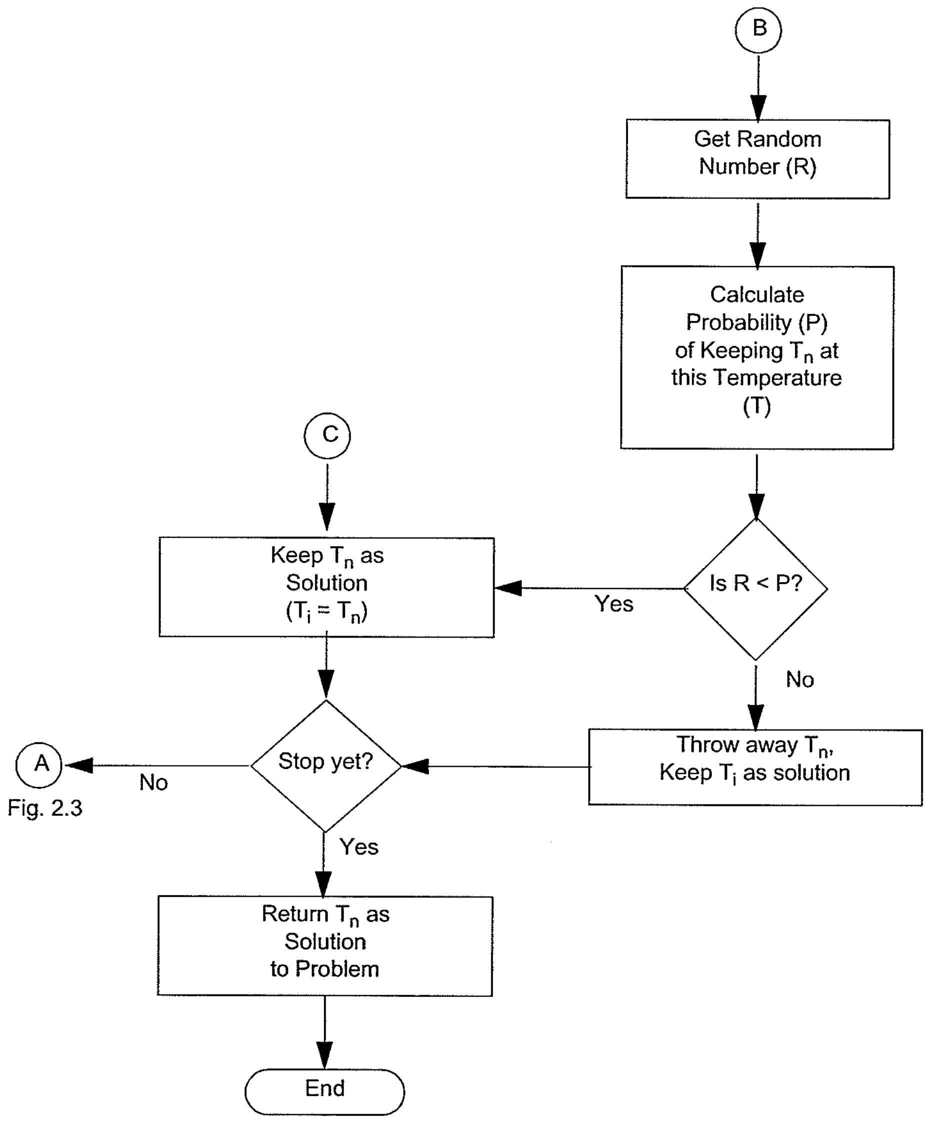

B
Get Random Number (R)
Calculate Probability (P) of Keeping T_n at this Temperature (T)
Is R < P?
Yes
No
C
Keep T_n as Solution ($T_i = T_n$)
Throw away T_n, Keep T_i as solution
Stop yet?
No
A
Fig. 2.3
Yes
Return T_n as Solution to Problem
End

Fig. 2.4 Flowchart of the SA algorithm - part 2.

an inferior solution is kept, it is critical that the temperature is calculated correctly. First of all, an initial temperature must be found that is sufficiently hot that most inferior solutions will be kept (typically we aim for 95% of inferior solutions being kept). Secondly, it is imperative that the cooling of the temperature be carried out in a controlled manner to ensure that quenching does not occur, and to increase the likelihood of achieving the global minimum. The most common cooling schedule for SA is shown in Fig. 2.5 and is the profile used in this thesis.

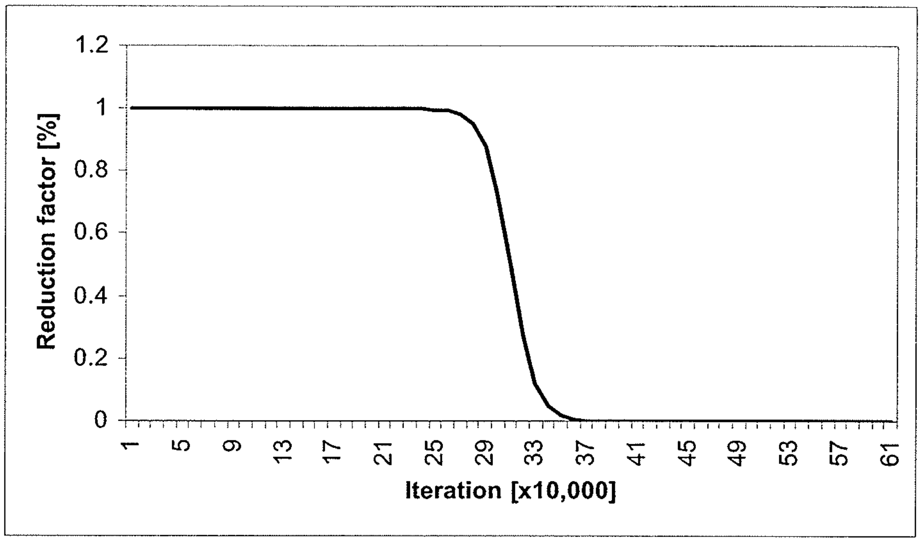


Fig. 2.5 Temperature cooling profile for SA.

Typically, we work in the order of thousands of iterations, and thus each tick in Fig. 2.5 can be viewed as 10,000 iterations. The cooling profile of Fig. 2.5 is highly effective, due to the three distinct phases of the function. Initially, the temperature is cooled very slowly to allow the optimization algorithm a chance to sample a wide range of solution across the solution space. In the intermediate stage, it is assumed that the algorithm has found a very good region to continue searching, thus the temperature decreases more quickly, encouraging the algorithm to climb down towards the minimum of the region. In the final stage, the temperature is once again cooled slowly to allow the algorithm to shuffle in the current solution region and find the absolute global minimum of the region

Perhaps the reader may still wonder why we would want to use SA and not a simpler approach, such as the steepest decent or some other non-deterministic algorithm. The answer lies in the randomness of the algorithm. Because the SA algorithm is allowed to search the entire solution space of an optimization problem randomly, it is possible to escape local minima in search for the global minimum. In many of the deterministic optimization approaches, the starting point of the algorithm completely dictates what the final solution will be. For further information on SA, consult the book by Aarts and Korst [AaKo89] and Section 2.2

2.5 Artificial Neural Networks (ANNs)

Note that the information contained in this section was extracted from the following sources [ChMu98] [Gins97] [Kasa96] [Mast93] [McRu88] [MeMR97] [RaRa95] [RuMc88] and [WeUS88]. Specific research cited is supported by specific references.

Artificial neural networks are computational models based loosely on biological neural processing that occurs in the brain. It is important to note that although ANNs have similarities to the human brain, they are not intended to model it directly, rather they are an attempt at solving problems in a "human like" manner. An ANN has two main components: (i) neurons that perform processing, and (ii) connections between the neurons that have associated weights (synaptic gaps). Signals pass from neuron to neuron along the connections, being multiplied by the connection strengths (weights) which can be either positive or negative. At the neurons, all of the input signals are summed and modified by a function (activation function) to calculate the output from that neuron.

2.5.1 History of ANNS

2.5.1.1 Initial Concept

The first mathematical model of a neuron is credited to McCulloch and Pitts in 1943. It was a very simplistic model with binary input and output, and a fixed activation level. However, this humble beginning lead to a rich research area that progressed at a steady pace. It took no time at all to discover that this new model was able to implement many arithmetic and logical operations. In 1949, Hebb made one of the most significant contributions when he demonstrated that a network of neurons could exhibit learning behavior when a learning law and repeated activation by other neurons were used. In 1954, Gabor introduced the learning law, which used gradient descent (as discussed in Section 2.2) to obtain “optimal” weights. These optimal weights minimized the mean squared error between the observed output signal generated and a desired signal. A crucial development occurred in 1961 when Rosenblatt proposed the initial backpropagation (BP) model (which is covered in detail in Section 2.5.7), which had a flaw of using non-differentiable activation functions (and was also a single layer model).

2.5.1.2 Near Demise

In 1969 a fatal blow was dealt to the research on ANNs in a book by Minsky and Papert [MiPa69]. It identified the incapability of the single layer networks in use at that time to solve many simple problems (in particular the XOR function). This finding demonstrated that ANNs were non computationally universal, resulting in a drastic and immediate reduction in research (and funding for research).

2.5.1.3 Rebirth

Fortunately, due to the perseverance of a few researchers, the topic of ANNs was resurrected after a near two decade drought. There were a number of workarounds identified by these researchers to address the shortcomings of the early ANNs, including: (i) adding more than a single layer of neurons; (ii) using learning laws other than gradient descent (which is not always successful in finding a solution) such as Boltzman machines and other stochastic methods. Theoretical methods of determining the capabilities of networks were developed and finally, hybrid systems were developed. These ANN techniques were introduced during the 1980s, and the 1990s produced many useful results.

2.5.2 Overview of Biological Neural Processing

It is estimated that there are approximately 10^{11} neurons in the human brain, with 10^{15} connections between them. The processing rate of the human brain is quite slow with only 10^4 operations per second (kHz) as compared to the latest desktop processing of

10^9 operations per second (GHz). However, the extremely parallel structure of the brain makes up for this lack of speed [MeMR97].

A typical representation of the neural process in the brain is shown in Fig. 2.6 [MeMR97]. The neurons in the brain are what perform the actual processing. The dendrites carry the signal into the neuron, and the axons carry the signal away from the neuron. These axons have many branches that connect to the dendrite of other neurons. The synaptic gap is situated between the axon and dendrite. For the signal to pass from the axon to the dendrite, the algebraic sum of the signals received must surpass a threshold value. If this is the case, a signal is generated on the dendrite and carried on to the cell body.

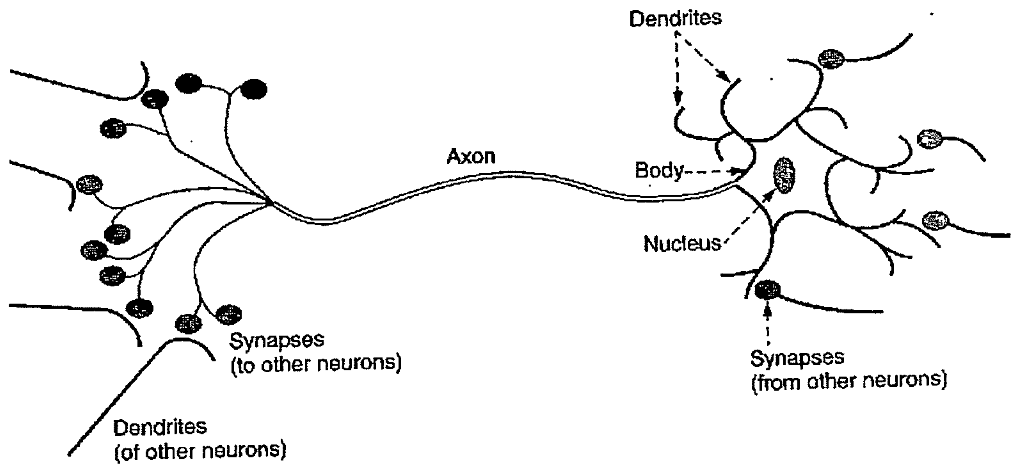


Fig. 2.6 Neural processing in biological brain.

Two of the major operations of the brain are learning (storing knowledge) and recall (using stored knowledge). When a brain "learns", it stores the information by changing the chemicals in the synaptic gaps. During recall, the synaptic gaps fire based on these chemicals. Thus the brain is able to produce an output for a new situation by generalizing what has already been learned. It is this property of the brain that we wish to model with ANNs.

2.5.3 Overview of ANN Processing

In ANNs, we emulate the function of the biological neuron as nodes and the synaptic gaps as weights on the connections between the nodes. A simple illustration of a node is shown in Fig. 2.7.

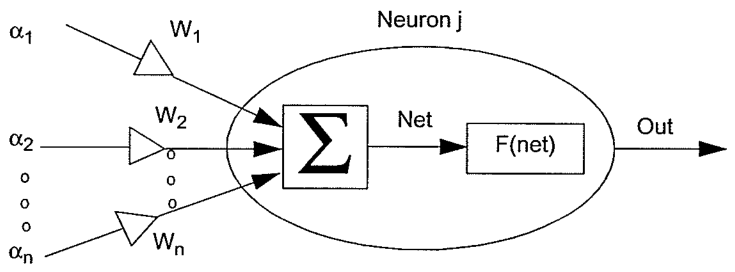


Fig. 2.7 Components of an artificial neuron.

Each neuron j has n inputs α_i, each passed to the neuron through a connection has a connection strength (weight) w_i associated with it, which modifies the signal from the originating source. If the weight is positive, it is said to be an excitatory signal, whereas if a weight is negative, we say that it is inhibitory. In order to find the output from a neuron, the n inputs to the node (α_i) are multiplied by their connection weights (w_i), and then summed (referred to as the Net input) according to

$$\gamma = \sum_{i=0}^{n} \alpha_i w_i \tag{2.2}$$

where γ is the dependent variable in the function (referred to as the activation function) used to generate the output value for the node. Numerous activation functions may be used in a neural network (such as the linear, threshold, Gaussian, and sigmoid functions). However, the sigmoid activation function (which has an activation level closest to that of the biological neuron) is the most common and will be used in this thesis. The sigmoid activation function (ζ) also called the squashing function is calculated according to

$$\zeta = F(\gamma) = \frac{1}{1+e^{-\gamma}} \tag{2.3}$$

The form of the sigma curve is shown in Fig. 2.8.

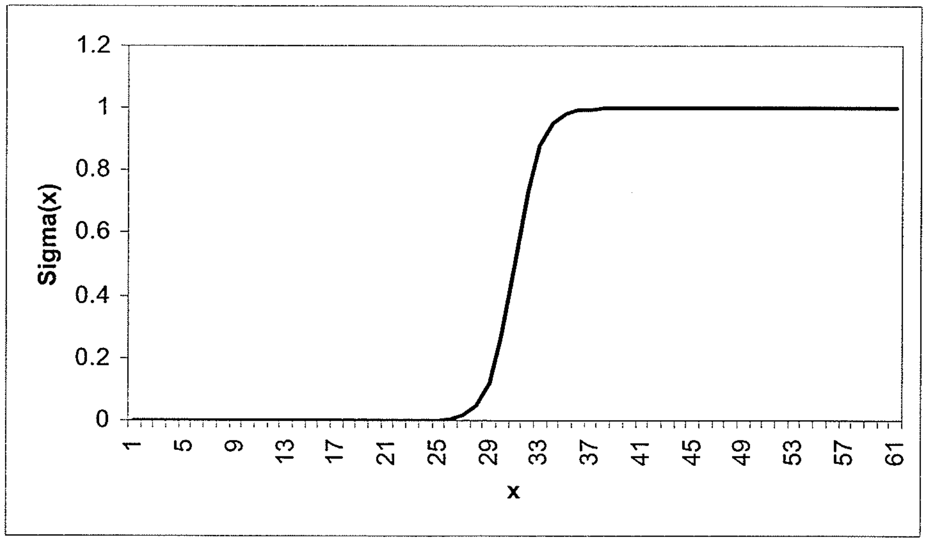


Fig. 2.8 Sigmoid activation function.

We can put a number of nodes together to create a network having far greater processing power than a single node. An example of a simple ANN is shown in Fig. 2.9, which has an input layer, a single hidden layer, and an output layer.

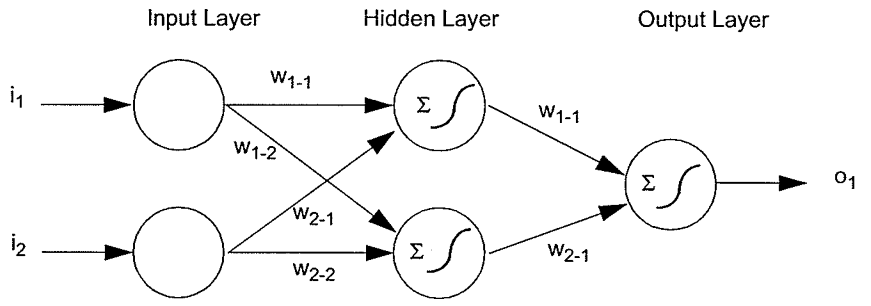


Fig. 2.9 Simple ANN.

The input connections have a weight of +1 (thus they do not affect the input signals, and the input nodes simply pass the signal they receive straight through without processing. Similarly, the output connections also have a weight of +1 so the signal generated by a neuron in the output layer is the output. It should be noted that ANNs are much smaller than the real neural process and are a simplification of the biological neural process.

2.5.4 Processing with ANNs

Similar to the biological process of acquiring knowledge, ANNs have two modes of operation: training (learning) and recall (applying what has been learnt). Although in most ANNs these are separate serial processes, there are some models that continue to learn while they are being used. The training process is quite straightforward for simple ANNs and can become as complex as the designer feels necessary for the larger, more complex ANNs. In the typical training process, the designer presents the network with sample data for which we have a known desired output. The ANN will generate an output that can be compared to the desired output. If the two do not match, then we can change the weights of the interconnections so that the actual output is closer to the desired output. How we change the weights depends on the learning rule that we are using. The most common learning rule is backpropagation (BP), which is a form of gradient descent optimization of the weights. The error between the desired and actual output is propagated back through the network, assigning blame to interconnections based on their contribution to the output, and adjusting the weights accordingly (as explained further in Section 2.5.9). In this manner, the ANN acts as a memory, which internalizes all of the data it is presented with during the training process, so that it can generate output during the recall process.

When the ANN is used for recall, it is presented with the pattern for the desired output, and generates an output for it. The correctness of the output depends on many factors, the most important of which is how well the network was trained.

2.5.5 Why Use an ANN?

The question may arise that since there are many methods of optimizing problems, some of which have been identified in this thesis (Section 2.2), why would one want to use an ANN to optimize brachytherapy treatments? For difficult problems (computationally expensive) we must resort to non-deterministic methods of optimization such as the GA or

SA. The major drawback with these methods is the excessive length of time required to generate an output. Therefore, some would argue that we turn to the ANN simply for the speed with which it can generate an output (due to its massively parallel structure). However, most researchers who understand ANNs know that this is not the only appeal of ANNs: they have the ability to generalize and create correct answers for inputs which have never been seen before.

2.5.6 Types of ANNs

There are many different types of ANNs, and they can be classified according to the way they are organized (the activation function of the nodes and way the nodes are connected). Different training techniques can also distinguish one type of ANN from another, but typically the organization is the distinguishing feature. For example, a classical Hopfield ANN [MeMR97] is a fully connected network, with every node being connected to one another. The inputs of the Hopfield network are used to excite the network, by giving the nodes an initial state. The Hopfield network then cycles until it settles on an output. It is very much like a non-deterministic optimization algorithm searching the solution space for a minimum. Clearly this ANN is extremely different from the BP ANN, yet they are both considered ANNs.

Some ANNs are combinations of two or more types of ANNs and are called hetero-associative networks. Perhaps the most common hetero-associative network is the counter-propagation (CP) network. It is composed of two layers: the Kohonen and the

Grossberg layers. The Kohonen layer finds the nearest neighbour (closest match in its memory bank) of the input pattern, which is then used to select the set of weights for the Grossberg layer (which is very similar to a BP ANN).

The radial basis function (RBF) ANNs use two different learning rules during training. This enables a RBF ANN to learn local features yet still be able to interpolate outputs for new unfamiliar inputs (to a limited extent). The RBF ANN is similar to the BP ANN except that the BP learns global features, enabling it to generalize better than the RBF ANN, without the need for two training phases.

There is an infinite number of ANNs, as we can come up with an infinite number of combinations of connection schemes, activation functions, and learning rules. The focus of this thesis is to identify a scheme of optimizing brachytherapy treatments with an ANN. The ANN model used in this thesis is the BP ANN as it is the ANN model with the most extensive background. It should be kept in mind that perhaps some of the best ANN designs are yet to come and there may be an ANN perfectly suited to the brachytherapy optimization problem that has yet to be found.

2.5.7 BP ANNs

BP is the most common form of ANN used to date. It was initially introduced by Rosenblatt in 1961 but had the fatal flaw of not using a differentiable activation function, thus limiting what it was able to learn. Better learning algorithms became available as early as 1962 (Dreyfus), 1969 (Bryson and Ho), and 1974 (Werbos). But it was not until the work of McClelland and Rumelhart (1986) that the BP algorithm, as we know it, became popular.

2.5.8 BP Organization

Although the BP ANN has a very well defined form, there are some variations from the standard form of the BP ANN but these will not be covered in this thesis. This thesis uses the classical form of the BP ANN. The BP ANN is a feed-forward network (information flows from the inputs to the outputs). It has an input layer, one or more hidden layers, and an output layer. Processing proceeds sequentially from the input nodes, through the hidden layers, and on to the output nodes. Every layer is fully connected to the next layer. Thus the output from a node in layer j is connected to the input of every node in layer k. Although there can be many layers, typically only three layers are used because the additional layers drastically increase complexity (in terms of analyzing the functioning of the inner layers, as well as the time required to train the network, which is exponentially tied to the number of inner layers) with little performance gain.

2.5.9 BP Training

The purpose of training an ANN is to adjust the internal weights so that when specific inputs are applied, specific outputs are generated. In order to train the BP ANN, we need to have a set of training data consisting of input vectors with their corresponding desired output vectors. This input and desired output vector set is called a training pair.

Typically we need a number of training pairs called a training set. The number of training pairs included in a training set is problem specific.

To adjust the weights in the ANN using the training set, requires a training algorithm. The BP ANN gets its name from its training algorithm, although a more accurate name is *error backpropagation* not just *backpropagation*. Error is back projected from the output nodes back through the hidden layers towards the input nodes. The activation function of the nodes is sigmoidal (and therefore differentiable everywhere, a necessity for the BP algorithm as it uses the derivative during training). However, the sigmoid activation function also has the added benefit of gain control, meaning that large signals do not saturate the network. Before we start the training process, we initialize the weights in the network to small random numbers to prevent saturation (large weight values). By doing this, we also prevent finding the same local minimum (which occurs with constant static initial values), and paralysis (a state in which training has no effect on the network because the small weight change is negligible with respect to the large weights).

The training process uses the following five steps:

1) Select a training pair (X,T) from the training set and apply the input vector X;

2) Calculate the output vector Y;

3) Calculate the error between Y and the desired output T (from training pair);

4) Adjust the weights in the network in order to minimize the calculated error; and

5) Repeat Steps 1 to 4 until a desired error level is achieved.

Note that when the BP ANN is used for recall, only Steps 1 and 2 are used.

Although it is not apparent, the identified steps are actually performed in two passes. Steps 1 and 2 are part of the forward pass, whereas steps 3 and 4 are part of the backward pass.

The forward pass begins with the application of the input vector X in which the vector is multiplied by the weights between the input layer and the first layer. At each node in the first layer we sum these products and use the resultant in the activation function to produce an output. Once this is repeated for each node in the layer, the processing continues to the next layer. This process is repeated until we arrive at the output layer, which simply produces the output vector Y as the summed inputs from the previous layer (γ) passed through the activation function ($Y = \zeta(\gamma)$).

The backward pass is used to change the weights in the network, to bring the outputs closer to the desired value. Adjusting the weights for the output layer is straightforward, as we can calculate the error at the output based on the difference between the produced output vector Y and the target output vector T (provided in the training pair). The amount required to change the weight between the output node and the hidden layer before it, is calculated according to

$$\delta = (T - Y)\frac{\mathrm{d}}{\mathrm{d}x}\zeta(\gamma) \tag{2.4}$$

where $\frac{\mathrm{d}}{\mathrm{d}x}\zeta(\gamma)$ is the derivative of the sigmoid activation function (ζ) which can be simplified as

$$\frac{d}{d\gamma}\zeta(\gamma) = Y(1-Y) \tag{2.5}$$

and thus Eq. 2.4 reduces to Eq. 2.6

$$\delta = (T-Y)(Y(1-Y)) \tag{2.6}$$

Next, we multiply Eq. 2.6 by the learning rate (η) which is a value that is used to control how much of an effect the error will have on the weight chance, and the output from the node in the hidden layer (Y_h) to this particular node in the output layer, which is an attempt to "assign the blame" for the error as shown in Eq. 2.7.

$$\Delta w = \eta\delta Y_h \tag{2.7}$$

The new weight between the nodes is then calculated according to

$$w(n+1) = w(n) + \Delta w \tag{2.8}$$

This entire process is a version of a gradient descent optimization on the weights which is why we use the derivative of ζ in the weight change.

Changing the weights for the inner layers (also called hidden layers because they have no associated target vector) is more difficult than in the output layer. Thankfully the work of Rumelhart *et. al.* [RuMc88] provides a solution that involves propagating the error backwards using the weights that were used to generate the output. Equations 2.7 and 2.8 are used to change the weights in all of the layers, however, the value of δ is different for the hidden layers. The δ is known for the output layer (using Y and T), but it must be found for the hidden layers without the benefit of having a corresponding target vector (T). The solution to this problem is to construct a special δ value for the hidden nodes by back propagating the δ values from all of the nodes in the following layer. The δ values from the following layer are weighted using the weights that were applied to the output from the current layer, thus assigning more blame to connections that have larger weight and δ values. The process is shown in Fig. 2.10.

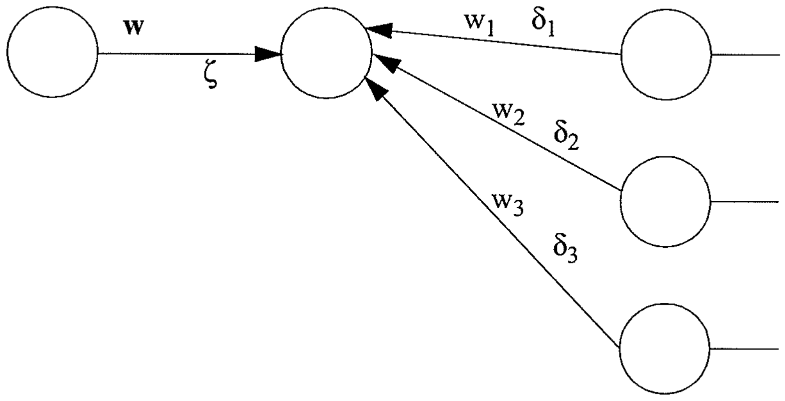


Fig. 2.10 Training a weight (w) in a hidden layer.

The equation used for the calculation of δ for internal layers assuming a sigmoid activation function is shown in Eq. 2.9. δ_i is the δ value of the i^{th} node in the following layer and w_i is the weight connecting the node in question with the i^{th} node in the following layer.

$$\delta = \zeta(1-\zeta)\left(\sum_i \delta_i w_i\right) \tag{2.9}$$

In order to update the weights for a layer, the δ values from the following layer must be known first. Therefore, we start at the output layer and progress towards the input layer, hence the name *backward* propagation.

There are two techniques of updating the weight changes in an ANN. They are referred to as on-line or batch (off-line). In on-line training changes in the weights are applied immediately after they are calculated, thus using the newly calculated weights on the next training pair. In batch training the weight changes are accumulated through the entire training epoch (complete set of training pairs) and only applied as an accumulated weight change at the end of the epoch. There are valid arguments for both methods of training, the main points being that batch training tends to be faster but requires additional storage space. Due to the large sized ANN developed for this thesis, it was decided to use an on-line approach to minimize additional storage space.

A bias term can be included in the neuron that serves as a shift of the origin of the activation function along the horizontal axis. This has a similar effect to adjusting the threshold of the neuron, thereby permitting a more rapid convergence of the training process. The effect of a bias is shown in Fig. 2.11. It was decided not to include a bias in this thesis work as the same resultant ANN will be achieved if the training is performed without the shift, albeit much slower.

Another scheme of improving the training time is to use a momentum term that combines the previous weight change with the new weight change. Thus, a weight change in the same direction becomes larger whereas a weight change in a different direction will be dampened, thus preventing oscillation of the weight around the actual minimum. The same resultant ANN will be achieved without momentum, but more slowly. No momentum term was used in this thesis as the same resultant ANN is achieved without shifting sigma, albeit possibly slower.

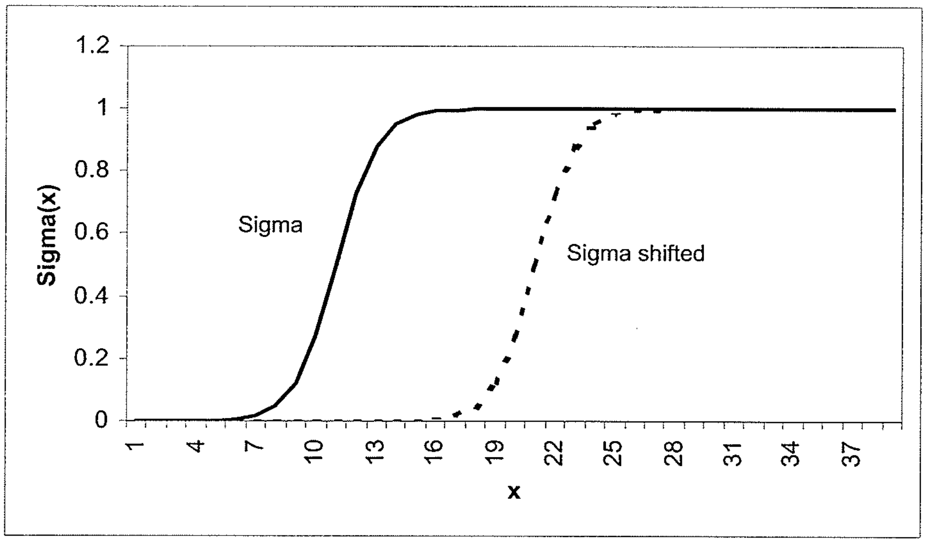


Fig. 2.11 Effect on activation function when adding a bias.

2.5.9.1 Additive Noise in the Training Data

In order to increase the robustness of the performance of an ANN we can introduce small random values (noise) into the training data. The noise is injected into the desired output vectors of the training data [WaPr99][Kasa96]. This has two effects: (i) to improve the ability of the network to generalize, and (ii) to increase the speed of training and its ability to escape local minimum. Additive noise was used in this thesis, as the amount of training data was small; and adding noise to the desired output vectors is in some ways equivalent to adding more training sets.

2.5.9.2 Cooling η During Training

In a manner similar to SA, η can be cooled during the training process to achieve a better ANN. The purpose for cooling η is identical to the cooling of the temperature in SA. It allows the training algorithm an opportunity to search the solution space in an effort to find the global minimum. In the initial phase of training, a large η value causes large and drastic weight changes in the ANN, thus allowing it to jump around the solution space. As the training process continues, η decreases and eventually the weight changes become very small as the ANN settles into a minimum (hopefully the global minimum). The software developed in this thesis employs this technique during the training of the ANN.

2.5.10 ANN Verification and Validation

When the training of the network is finished, an associated final error rate on the training data is obtained. This error rate is referred to as the *apparent error rate.* However we would really like to know the *true error rate* of the network on an arbitrary data set, not just the training data. Unfortunately, for most problems there are many (if not an infinite number of) patterns that may be presented to the network, therefore making it is impossible to test them all. Thus a technique for assessing the true error rate is necessary. This process is referred to as verification or validation.

There is a number of techniques of validating a network. Usually the deciding factor in choosing a techniques is the number of training patterns. The method used in this thesis is called *holdout processing*, and involves separating the training data into two sets. The network is trained using one set, and then verified using the second set. Other schemes such as k-fold cross validation involve dividing the training data into k partitions, using all of the partitions but one to train the network, and then verifying the network with the partition that was held back. This process is then repeated k times holding back a different partition, and the true error rate is reported as the average of all of the true error rates. Certainly k-fold cross validation is a more robust verification method, but we have too few training data to employ it in this thesis.

2.5.11 Choosing the Best ANN

Although the design of an ANN is a scientific process, it also has heuristic components. The scientific process involves defining what features the network should learn, deciding on a learning strategy, and preparing the data for training. The more heuristic process (although still science) is defining the ANN architecture and its structure. This is a fundamental step in the development of an ANN and requires experience and knowledge in order to make correct choices. A portion of the experimental results and discussion of this thesis involve finding the best ANN architecture for the brachytherapy optimization problem.

2.6 Chapter Summary

All of the background information required for the concepts utilized in this thesis were presented in this chapter. The focus was on brachytherapy, SA, and ANNs. Using the information from this chapter, we designed a system to optimize brachytherapy implants. The design of the system is presented in the next chapter.

CHAPTER III

SOFTWARE REQUIREMENTS

"I must create a system, or be enslaved by another man's."
- William Blake (1757-1827)
- British poet, artist, The Marriage of Heaven and Hell

"The principal goal of education is to create men who are capable of doing new things, not simply of repeating what other generations have done - men who are creative, inventive and discoverers."
- Jean Piaget (1896-1980)
- Swiss child psychologist, noted for cognitive development in children

3.1 Software Overview

The software system developed in this thesis is used to find optimal source positions within 2D square tumours using an ANN, with the goal to prove that ANNs can be used to optimize brachytherapy implants. However, before software can be developed a design is required, which must be based on the requirements that the software must meet. This chapter will identify the software requirements. The brachytherapy optimization problem is broken into three separate steps: (i) the development of data to train the ANN with, (ii) the actual training of the ANN, and (iii) using the ANN. The following sections will identify the requirements for each of these three steps.

3.2 Requirements for Software Used to Create Training Data

The training data is created using the SA process as described in Section 2.4. A program is developed and given the name simulated annealing for brachytherapy (SAB).

The SAB software must be able to load a tumour shape from a file and use SA to find optimal source positions within tumour. The software must be able to report the output to the user in a meaningful and useful manner. The software also must be controllable by the user in order to adjust the functioning of the SA. For instance, the software must have an option to allow sources to be placed anywhere in 2D space, or to be confined only to the tumour. There must be an option to enable or disable the use of variable dwell times for the sources. There must also be a set of parameters that can be used to control the SA algorithm, including a maximum number of iterations at each temperature value, an initial acceptance ratio and a cooling schedule. The ratio (or resolution) of the input tumour in terms of pixels per millimeter to the SAB software is a parameter that is required as input. The input to the SAB software and the output it produces will be used as a training pair to train an ANN. The requirements for the software used to train the ANN are covered in the following section.

3.3 Requirements for Software Used to Train an ANN

The training pairs developed with the SAB software (Section 3.2) are used to train the ANN. The software that is developed for this purpose is a malleable ANN (MANN) that can use any form of input and desired output to adjust the weights in the layers of the ANN using the BP algorithm. The MANN software should be very flexible in order to accommodate any form of input and output investigated. In other words, the size of the input and output layers of the ANN can change dynamically The number of hidden nodes in a single hidden layer (by design the ANN developed will only use a single hidden layer) must also be variable. The MANN software must treat the stopping conditions and the rate of training as input variables, so that different training methods may be investigated. Finally, MANN must be able to start training from a previously trained set of weights, or from a new set of random weights, based on the user's preference. The requirements for the software that will use the ANN developed with MANN will be covered in the following section.

3.4 Requirements for Software that Uses an ANN for Optimization

The software used must be able to load tumour shapes from files and allow a user to create sample tumour shapes. The program must use an ANN to find optimal source positions in the tumour shapes and be interactive, to allow the placement of sources by a user to simulate sources being inserted in the patient. Finally, the software must provide tools for the user to evaluate the optimized plans (such as isodose distributions in 2D and 3D) and provide statistics on the current source configuration. A software program will be developed and given the name Brachytherapy Optimization With Artificial Neural Networks (BowANN).

3.5 Chapter Summary

The SAB software will be used to create training pairs to train an ANN. The MANN software will be used to train an ANN. Finally the BowANN software will be used to interface the brachytherapy optimization ANN to a user. In the following chapter, the requirements for each of the three programs is used to implement the software.

CHAPTER IV

SOFTWARE IMPLEMENTATION

"Try not! Do, or do not. There is no try."
- Jedi Master Yoda

4.1 Introduction

In Chapter 3 the requirements for the software that are used in this thesis were covered. These requirements are used to map the software to hardware. Once the hardware is mapped, the software is designed and implemented. This chapter covers the mapping of the software to hardware and the implementation details for the software.

4.2 Technology Mapping

Before the software is implemented, the hardware system for which it is targeted must be decided. As this research does not have equipment funding, only existing hardware can be utilized. A number of considerations where made in deciding which hardware should be used for which software.

4.2.1 Simulated Annealing for Brachytherapy (SAB)

The first component to be developed is the SA aspect of the overall system. As the execution time for SA can be lengthy, the target hardware should have as fast a processor (CPU) as possible. Another consideration is the ability to run the software on a number of different machines at the same time (a form of parallel computing), which obviously would increase throughput. As the University of Manitoba (UM) has a number of *open-area* computing facilities with Sun SPARC and ULTRA workstations running the Solaris Operating System (OS), it was decided that SA would be implemented to take advantage of these resources. The fastest computer at the time of development that was available at CancerCare Manitoba (CCMB) was a Silicon Graphics (SGI) O2 with a MIPS R10000 CPU running the IRIX OS. For these reasons, the SA component was implemented for the UNIX platform, as IRIX and Solaris are both based on the POSIX OS. This means software developed for POSIX can run on all of the open area machines at UM and on the SGI at CCMB.

The next consideration for SA was the computer language to be utilized. As the hardware was to utilize the SGI O2 as well as the Sun SPARCs and ULTRAs, which run IRIX and Solaris respectively, the code had to be able to be compiled for both OSs. The only compilers in common between the two OSs where FORTRAN and C. C was chosen as it is most familiar.

4.2.2 Malleable Artificial Neural Network (MANN)

An ANN can be developed in software or in hardware. In hardware problem specific hardware can be developed or reconfigurable hardware that will allow for the experimentation of the ANN layout can be used. However, due to budgetary constraints on this thesis a software approach was taken. When the time came to develop the ANN software, a new computer had been purchased at CCMB for another project. The computer is an Intergraph TDZ2000 GX1 workstation, which has 512 MB of RAM, an Intel PIII 500 MHz Xeon CPU, and has the Microsoft Windows NT OS. This machine has a lot of power (in the CPU speed) and memory, both of which are useful for the learning and recall components of the ANN. As the Intergraph workstation is running the Windows NT OS, the ANN development had to take place for the Windows platform. It was decided to keep the code portable to Windows 95 and 98, so the development was not to use any Windows NT specific function calls. The available language choices were vast on the PC platform, but the structure of ANNs is very object oriented (OO) in nature, and thus a language capable of OO development was desirable. The language that was chosen was Microsoft Visual C++, which is the standard for Windows development and has the OO capabilities that were required for MANN.

4.2.3 Brachytherapy Optimization with Artificial Neural Networks (BowANN)

The final stage of development is the creation of the program that uses the ANN trained in MANN to optimize source positions in the 2D tumours. A similar logic to that used for the hardware mapping of MANN was utilized in choosing the Intergraph TDZ2000 GX1 workstation, as well as Microsoft Visual C++ for the development of BowANN.

4.3 Software Implementation

Now that the software has been overviewed (Chapter 3) and the hardware that is used to create it has also been specified (Section 4.2), the implementation details of each of the three programs, SAB, MANN, and BowANN is covered.

4.3.1 SAB

The form of the SA algorithm for this thesis is as shown in the flow diagram of Fig. 4.1. The addition and deletion of sources has a very dramatic effect on the cost function and therefore must be done in a controlled fashion. It was decided that SAB would use the cost function and SA to find the best solution for a given number of sources. Then, another source would be added and the SA would run again on this new number of sources. This process continues until a solution is found that satisfies the cost function exactly or a maximum number of sources has been added. SAB stores the best solution encountered over all of the different number of source counts, and this is reported as the final result. The reason for this is that by continually adding sources better solutions are not necessarily being generated. Therefore, the final output from the SA should not be just the last simulation that was carried out, rather it should be the best solution obtained over all of the source counts.

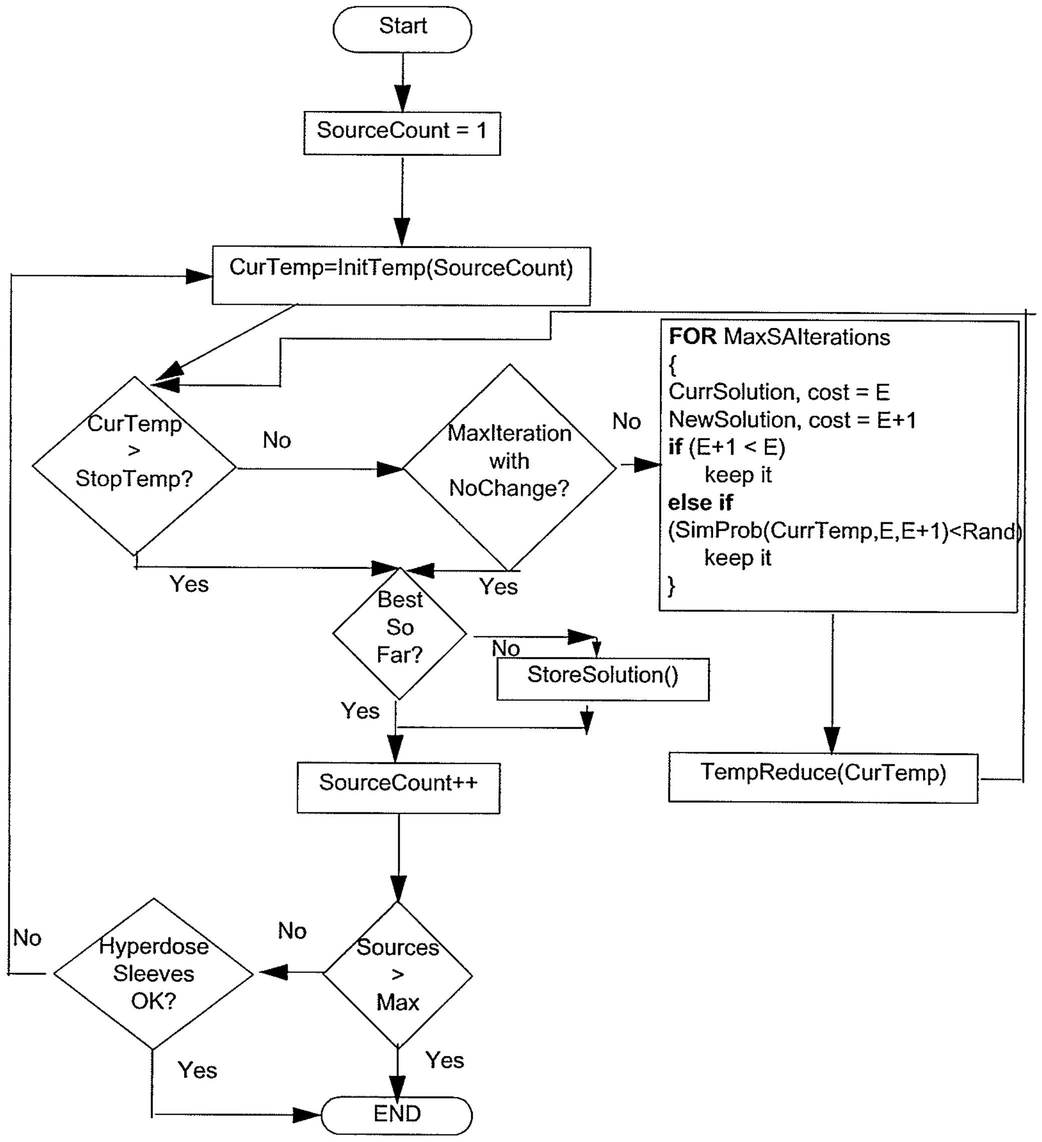


Fig. 4.1 Software flowchart for simulated annealing.

4.3.1.1 SAB Input

As identified in Section 4.2.1, the SA software is designed for UNIX, using C as the programming language. As tools to facilitate the creation of a graphical user interface (GUI) for UNIX were not available, it was decided that a text-based approach would be used for the user interface (UI). The input to the SA algorithm is a tumour shape to be optimized and a configuration file that sets the default and initial parameters for the simulation.

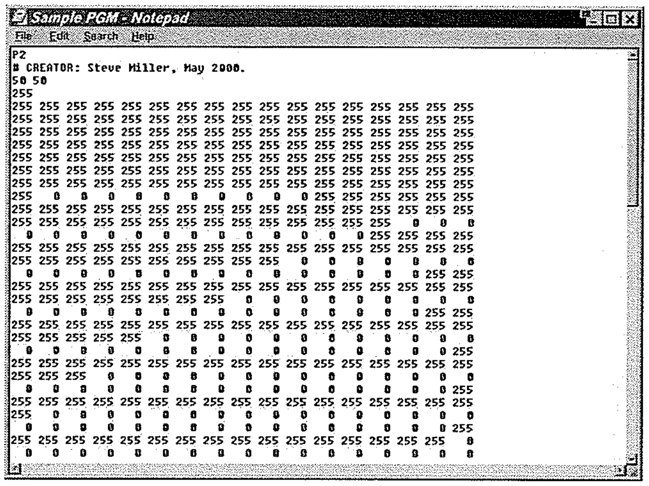
The form of the tumour file is of the *pgm* standard, which is a text-based digital image format (as opposed to a binary based format). A text based format was used to simplify the modification and reading of the file. There is a header which identifies the file as a *pgm* file (version 2), a comment line to insert identification remarks, the *x* and *y* size of the image, a number which indicates the number of color levels in the image, and finally, the individual pixel values expressed as integers (17 on each line). Fig. 4.2 shows the composition of a typical *pgm* file.

For this thesis four gray scale values are used to differentiate the elements in an image as shown in Table 4.1 and Fig. 4.3

Table 4.1: Grey scale values for pgm file representation.

Grey Scale Value	Element
0	Tumour
40	Periphery
175	Point Source
255	External

In future work the addition of critical structures will be implemented and will require an additional gray scale value. The graphical representation of the tumour *pgm* file of Fig. 4.2 is shown in Fig. 4.3 with the gray scale values identified.



```
P2
# CREATOR: Steve Miller, May 2000.
50 50
255
255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
255   0   0   0   0   0   0   0   0   0   0 255 255 255 255 255 255
255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
255 255 255 255 255 255 255 255 255 255 255 255 255 255   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   0 255 255 255 255
255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
255 255 255 255 255 255 255 255 255 255   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 255 255
255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
255 255 255 255 255 255 255 255   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 255 255
255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
255 255 255 255 255   0   0   0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 255
255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
255 255 255   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 255
255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
255   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 255
255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255   0
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
```

Fig. 4.2 Format of PGM files for SAB.

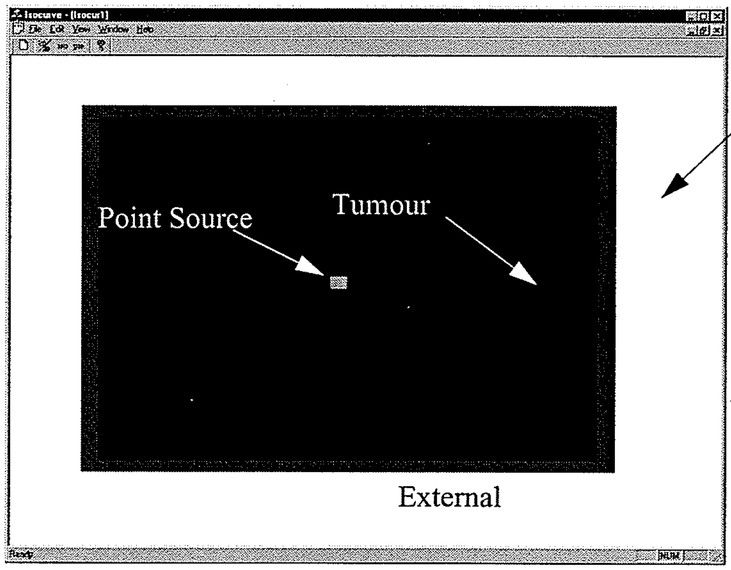


Fig. 4.3 Typical *pgm* file used in SAB.

4.3.1.2 SAB Configuration Parameters

There are many parameters that can be specified in the configuration file, however some are much more important than others. The information in the configuration file is used to meet the requirements identified in Section 3.2. The most important parameters are discussed first.

The *trial number* is an integer that identifies the current trial run and is used to store output information in files. For example to create the cooling profile output for trial "10" the output file would be called cooling-10.dat. If a unique number is not specified, past output will be overwritten.

The *maximum iterations* is another crucial parameter which specifies the number of iterations at each temperature in the SA process. Once the maximum iterations is reached, the temperature is cooled according to the cooling profile. Experimentation is required for each specific tumour size and shape to determine what works the best. It is not necessary to experiment if time is not an issue as a very large number will ensure the optimal result is found. If the algorithm has a sufficient number of iterations to minimize the cost at a particular temperature, the optimal solution will be found. Unfortunately, time is almost always an issue, and in order to reduce the amount of time spent by the SA process, the number of iteration at each temperature should be restricted and this must be done without compromising the quality of the solution.

The next parameter of importance is the *scale* or *ratio* of the input tumour file. This value informs the SA algorithm how to interpret the input file dimensions. It is an integer value that represents how many pixels in the image are equivalent one physical millimeter. For example a ratio of 3 would indicate that a 30 pixel dimension of a tumour is 10 mm or 1 cm in physical measurement.

The last group of parameters with significant importance are a number of flags that control the flow of the algorithm. The first flag sets up the algorithm to move sources either exclusively without altering the number of sources, or to add and delete sources automatically. It was decided that the addition and deletion of sources was too profound on the cost function and therefore, the flag is set to just move sources, however, future work could involve modifying the algorithm to be more dynamic. Next is a flag that would allow there to be no sources as a solution. This flag is set to false for the work presented in this thesis bust can be used in combination with the flag that allows the automatic addition and deletion of sources. Next is a flag that controls the location of the sources, by keeping them within the tumour, or allowing them to be placed anywhere. For the present research, this flag is set to restrict sources within the tumour volume. A flag is also specified to account for variable dwell time. If this flag is true, sources can occupy identical locations which in essence is identical to allowing the sources to possess different strengths or different treatment times. For the present research this flag is disabled thus restricting sources to be of equal dwell time (equal strength). Finally, a flag is specified which controls whether or not a heuristic will control the addition and deletion of sources. If this flag is true, sources are added until the hyperdose sleeves surrounding all sources are at most 1 cm^2 (as explained in Section 4.3.1.5). The work in this thesis set the heuristic flag to true.

4.3.1.3 Initial Temperature Calculation in SAB

There is also a set of parameters in the configuration file that affect how the initial temperature is calculated every time a source is added. There are three parameters that control the process, the first being the *maximum number of iterations* at each temperature. After the maximum number of iterations is reached, the temperature is increased by the *temperature increase factor*. Each time the temperature is increased, the number of sources being simulated are repeatedly added randomly. A calculation is used to determined how many times the new random solution is kept according to the cost function. The number of times the new random solution is kept is divided by the maximum number of iterations and defined as the *acceptance ratio*. This processes is repeated until an acceptance ration greater than the *initial acceptance ratio* is obtained. The pseudo code to calculate the initial temperature is as shown in Fig. 4.4. Essentially, the temperature is increased until the desired initial acceptance ratio is achieved. The temperature is initially low and a number of solutions are randomly generated. Then number of solutions accepted (according to our cost function) at that temperature are recorded. If the number is at least as large as the initial acceptance ratio, the temperature is returned as the starting temperature, otherwise the process is repeated with an increased temperature. This is an iterative process that is repeated until the desired acceptance level is met. The temperature found with this process is then used as the initial temperature for the SA.

```
Cost1 = InitialConfiguration
Cost2 = NewConfiguration
Delta = InitialConfiguration - NewConfiguration
CurTemp = StopTemp
done = 0
while (!done)
{
        for MaxInitTempIterations
        {
                InsertSources()
                if (Delta > 0)
                        M1++
                if (rand() < e^(Delta/CurTemp))
                        M2++
        }
        if (((M1 + M2)/MaxInitTempIterations) > InitialS)
                done = 1
        else
                CurTemp *= TempIncreaseFactor
}
```

Fig. 4.4 Pseudo code for initial temperature calculation.

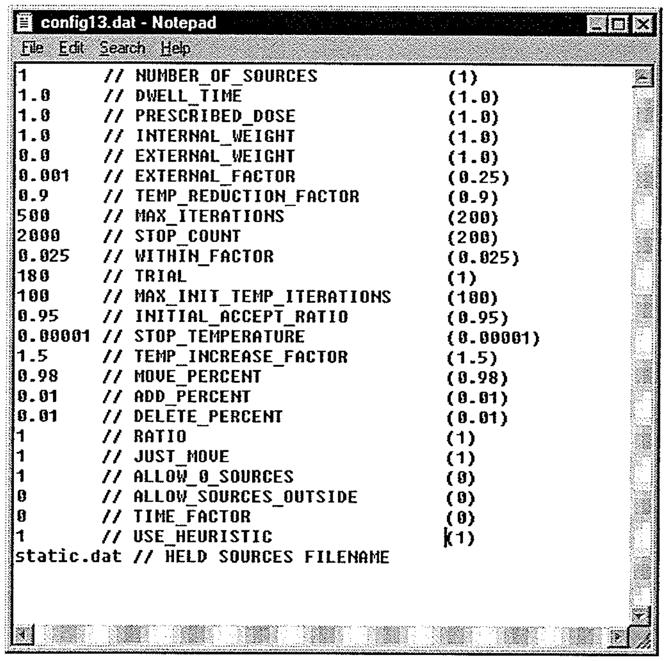
4.3.1.4 Other SAB Configuration Parameters

There are other parameters for SAB in the configuration file which are of less importance and thus not covered in as great depth as the previous parameters. This includes the number of sources with which the simulation should start. The SAB algorithm is monotonically increasing with respect to the number of sources. The SA will optimize using the starting number of sources, increase the number of source and then reoptimize. Typically, the starting number of sources is set to one so that SAB does not miss a situation in which the optimal solution has less sources than the starting number. Next is the default dwell time of the point sources. Although this thesis presents findings on static source times, the software is also capable of using variable dwell times, which is accomplished using this parameter. Finally, a filename can be specified in the configuration file that contains sources that can not be moved (static sources would represent a source that is already inserted in the patient and obviously can no longer be moved). Static sources were not used in the work presented in this thesis.

The form of the configuration file for the SAB is demonstrated in Fig. 4.5 (it should be noted that if these parameters are not specified, the algorithm will use the hard programmed defaults, identified in brackets).

4.3.1.5 SAB Cost Function Development

In order to evaluate the quality of a given solution a cost function must be utilized as identified in Section 2.4. The development of the cost function is crucial as it is the only means of assessing a given solution. In order to compare different solutions we evaluate dose as a percentage of the prescribed dose. To accomplish this, the solution is normalized to the minimum dose on the periphery (also known as the minimum peripheral dose and denoted by m_{PD}) of the tumour. By normalizing to this point all points on the periphery of the tumour receive at least 100% of the dose. This thesis will make the assumption that tumours are homogeneous for the sake of simplicity.



```
1          // NUMBER_OF_SOURCES             (1)
1.0        // DWELL_TIME                    (1.0)
1.0        // PRESCRIBED_DOSE               (1.0)
1.0        // INTERNAL_WEIGHT               (1.0)
0.0        // EXTERNAL_WEIGHT               (1.0)
0.001      // EXTERNAL_FACTOR               (0.25)
0.9        // TEMP_REDUCTION_FACTOR         (0.9)
500        // MAX_ITERATIONS                (200)
2000       // STOP_COUNT                    (200)
0.025      // WITHIN_FACTOR                 (0.025)
180        // TRIAL                         (1)
100        // MAX_INIT_TEMP_ITERATIONS      (100)
0.95       // INITIAL_ACCEPT_RATIO          (0.95)
0.00001 // STOP_TEMPERATURE                 (0.00001)
1.5        // TEMP_INCREASE_FACTOR          (1.5)
0.98       // MOVE_PERCENT                  (0.98)
0.01       // ADD_PERCENT                   (0.01)
0.01       // DELETE_PERCENT                (0.01)
1          // RATIO                         (1)
1          // JUST_MOVE                     (1)
1          // ALLOW_0_SOURCES               (0)
0          // ALLOW_SOURCES_OUTSIDE         (0)
0          // TIME_FACTOR                   (0)
1          // USE_HEURISTIC                 (1)
static.dat // HELD SOURCES FILENAME
```

Fig. 4.5 Configuration file for SAB.

For brachytherapy there are a number of issues to consider in the development of the cost function. The most significant is minimization of the size of hyperdose sleeves (areas receiving more than 200% of the dose), which from clinical experience should be restricted to cross sections less than 1 cm^2. If the hyperdose sleeve is larger, unacceptable tissue necrosis can occur. The hyperdose sleeve will increase as the distance between sources increases. A function that when minimized will ensure that hyperdose sleeves are kept within the constraints can be calculated according to

$$i = \left| \overline{\left(\frac{h_{CP}}{m_{PD}} \right)} - h_{max} \right| \qquad (4.1)$$

where h_{cp} are the hyperdose check points (points located 0.5 cm around every source in each orthogonal direction; the area contained between the four check points associated with any source will be 1 cm^2), m_{PD} is the minimum peripheral dose, (h_{cp}/m_{PD}) bar is the average dose at the h_{cp} points normalized to the m_{PD}, h_{max} is defined as 2 which corresponds to 200%, and the bars around the right side of the equation indicate the absolute value. As we are comparing the normalized dose at the check points to h_{max}, i will be a minimum when the hyperdose sleeves are less than 200%.

Another consideration during optimization is the attempt to maintain a homogeneous dose within the tumour. Areas receiving more dose stand a better chance at being killed, but risk tissue necrosis, whereas areas receiving less dose have less of a chance of being killed. However, it is impossible to obtain a homogenous dose distribution due to the significance of the non linear nature of the inverse square component of the dose calculation. Using a concept such as standard deviation to quantify the variation of dose in the tumour will report extremely poor results and is of limited use. As the dose is normalized to the m_{PD}, and since Eq. 4.1 ensures that the dose is not too high, it should be verified that there are no points in the tumour that are too cold (receiving less than 100% of the dose). The point in the tumour with the minimum dose (minimum tumour dose m_{TD}) is normalized to the m_{PD} and compared to 100% of the dose. A function that when minimized will ensure that all points within the tumour are receiving at least 100% can be calculated according to

$$j = d_{min} - \frac{m_{TD}}{m_{PD}} \tag{4.2}$$

where d_{min} is defined as 1 which represents 100%, m_{TD} is the minimum tumour dose and m_{PD} is the minimum peripheral dose.

Finally, it is critical to try and constrain the dose to the tumour, minimizing the dose to external structures, as radiation is destructive it is clearly desirable to avoid exposing healthy cells. It must be verified that the maximum dose on the periphery (maximum periphery dose M_{PD}) is not too high as that would indicate a large dose external to the tumour and possibly local critical structures. It was arbitrarily decided for this thesis that a dose of more than 105% on the periphery of the tumour would constitute too much dose. A function that when minimized will ensure that this condition is not violated is calculated according to

$$k = \frac{M_{PD}}{m_{PD}} - p_{max} \tag{4.3}$$

where M_{PD} is the maximum periphery dose, m_{PD} is the minimum periphery dose and p_{max} is defined as 1.05 which represents 105%.

There are a number of ways in which these functions (Eq. 4.1, Eq. 4.2, and Eq. 4.3) can be calculated and combined as demonstrated in research using similar cost functions [Slob92][YRPZ98][YuSc96]. However, for this thesis, it was decided to give the most importance to the size of the hyperdose sleeves, thus primarily using Eq. 4.1 as the cost function. The hyperdose sleeves help identify if the correct number of sources are being used. If the hyperdose sleeves are too big (indicated by h_{cp} being very large), more sources are needed. If the hyperdose sleeves are too small, too many sources are being used. Thus, if the addition of sources is controlled in SA and sources are only added once a good solution is found (using the current number of sources), the optimal number of sources will be found, as well as the optimal configuration of those sources. However, other factors in the cost function are required to monitor the other requirements for an optimized brachytherapy treatment. Thus, we include Equations 4.2 and 4.3, if they are evaluated to be positive, which may be viewed as *penalty factors*. If they are evaluated to be negative, this indicates that the requirement is met by the current solution. The final cost function is a combination of Eq. 4.1, Eq. 4.2, and Eq. 4.3 and is given by

$$c_{SA} = \begin{cases} i+j+k & \text{if (j>0),(k>0)} \\ i & \text{otherwise} \end{cases} \tag{4.4}$$

4.3.1.6 Source Movement in SAB

As was discussed in Section 4.3.1, sources are added in a controlled fashion. Sources are moved around within the tumour until the SA process reaches a termination condition, at which point (if all of the requirements are met) the SA terminates, otherwise, another source is added. The addition of the next source is actually like starting the SA process over again. There is no guarantee that the optimal solution for *n+1* sources is at all related to the configuration of *n* sources. Therefore, the sources are replaced randomly within the tumour and an initial temperature is recalculated and the SA process is then repeated.

Typically there is a single stopping condition for SA, however due to the complexity of this specific problem, it was decided to include a variety of dynamic stopping conditions related to the temperature and number of iterations. The first stopping condition occurs once the temperature drops below a value specified in the configuration file (absolute minimum). This is to prevent the algorithm from running if the probability of accepting a worse solution is close to 0. The second stopping condition occurs if a number of successive iterations have a cost value that is very similar. This is to prevent the SA from running for a long time, while there are no marked improvements occurring over long periods of simulating. This condition occurs not just at the global minimum but also if a local minimum is obtained and the temperature is not sufficiently high enough for the algorithm to escape. Finally, the SA will stop if the number of sources reaches an upper bound. This measure was taken for a number of reasons, the first being that it is not clinically viable to have an extremely large number of sources, and secondly, the algorithm has a hard time moving sources to free spaces if most of the spaces in the tumour are already occupied by other sources.

The movement of sources has three methods of operation depending on the parameters specified in the configuration file. The first consideration is how the source moves in general. The source movement is random with the possibility of movement in both the *x* and *y* direction. The pseudo code for the source movement algorithm is shown in Fig. 4.6.

The second condition that is considered in source movement is whether or not sources can be placed external to the tumour. If sources are allowed to be placed external to the tumour then no verification is necessary, otherwise, if a source gets moved external to the tumour the move is undone. The work in this thesis did not allow sources external to the tumour volume. The final condition that is checked when moving sources is whether or not dwell times are being considered. If dwell times are considered, then the sources are allowed to overlap, simulating a single source with an increased dwell time. The experiments presented in this thesis did not allow source overlap.

```
//x move
R1 = rand();
R2 = rand();
if (R1 < 0.33)
        x += movedistance
else if (R1 < 0.66)
        x -= movedistance
else
        x = x
//y move
if (R2 < 0.33)
        y += movedistance
else if (R2 < 0.66)
        y -= movedistance
else
        y = y
```

Fig. 4.6 Source movement algorithm in pseudo code.

4.3.1.7 SAB Output

SAB generates a number of outputs for analysis, and all of the outputs are stored to various files. First of all, SAB creates an actual visual picture of the solution achieved at every source number in the form of a *pgm* file. The *pgm* file is the tumour input to the SAB software with the optimized source positions superimposed on it. Another output that is reported is the cooling plot generated by SAB. The cooling plot should be a generally decreasing function and this provides a very simple and quick means of evaluating whether the SA algorithm is functioning properly. There are other output files that provide text-based feedback and report such things as the final source positions (the x and y coordinates), the execution time remaining in the current simulation, and special case stopping conditions among others.

4.3.1.8 SAB Random Number Generator

SA is a very long process requiring a large number of iterations in order to arrive at a solution. Each iteration uses a random number to determine the actions that the SA algorithm will take. Therefore, it is important that a good random number generator (RNG) be used. Typically a RNG is considered good if it can generate a long sequence of independent numbers. However, because a computer is a finite state machine, all RNGs implemented in software will be periodic. Some software RNGs are better than others. A thorough analysis of a few RNGs was recently conducted [Deni01]. The findings of [Deni01] are that the drand48 RNG (which has a period of 2^{48} - much larger than the number of iterations we anticipate doing in SAB) passes spectral tests. However, it was also found that the drand48 RNG failed the statistical test χ^2. This short coming of the drand48 RNG is overcome by ensuring that we only use the most significant 16 bits in the 48 bit value, as suggested by [Deni01]. The drand48 RNG will be used for SAB in spite of its lengthy execution time. The source code for the SAB software can be found in Appendix C.

4.3.2 MANN

The goal of the MANN software is to create an ANN with the ability to optimize source positions very quickly for brachytherapy cancer treatments. The MANN software must allow for a dynamic ANN structure as it is uncertain what the form of the ANN or its inputs and outputs will take. These will be determined through experimentation using the MANN software and by evaluating how well the ANN trained with MANN performs.

4.3.2.1 MANN Input

As stated in the background section on ANNs (Section 2.5) there are two methods of operation for an ANN, training and recall. Both stages have different input requirements. The following two sections will cover the differences in the inputs.

4.3.2.1.1 MANN Training Input

The ANN developed in this thesis will use the output from the SAB program (Section 3.2) as training data, and will use the BP method covered in Section 2.5.9 to train the ANN. The data for training must be formatted as textual input in a file. The first set of data in the file is an input vector to apply at the inputs of the ANN and the second set of data in the file is a desired output vector used to update the weights of the ANN. This training pair is the first of many that are stored in the file. There is a limit on the number of training pairs that MANN can handle (the maximum file size in Windows NT). This limit did not hinder the development of an ANN in this thesis. Once training is finished, the size of the layers and the weights that are found during training are stored to another text file. This file represents the knowledge that the ANN has acquired, and is used to load the ANN into the BowANN software.

4.3.2.1.2 MANN Recall Input

Recall in the BP network is similar to training however there is no BP of the error through the network. Therefore, the size of the layers and the interconnection weights can be loaded from the file created during training and used to generate output based on provided input. The input for recall could come from a file and the output generated can also be written to a file. The weights and layer sizes can be loaded into other software to incorporate the ANN into other programs such as BowANN. The ANN incorporated in BowANN can then use data from the current brachytherapy insertion simulation (RAM data) as input as opposed to using input from a file.

4.3.2.2 MANN Configuration Parameters

There are a number of parameters that must be specified before the MANN software before can be initiated. These parameters affect the structure of the ANN, as well as how the ANN is trained. The parameters are entered into a Windows dialog box when the MANN software is first invoked. Prior to the creation of MANN it was decided that only three layers would be used in the ANN, as adding additional layers increases the complexity. Therefore, it would be more advantageous to use a different ANN structure than BP to increase the complexity (most likely the probabilistic neural network (PNN)).

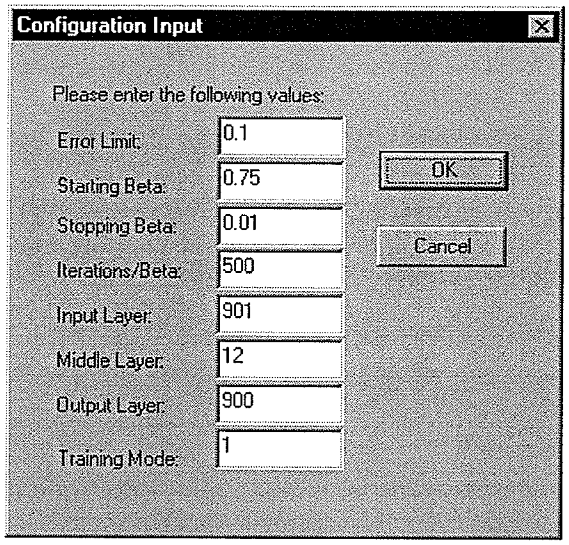
The first set of parameters that must be specified for MANN are related to the error in the system and the learning parameter η. First of all an error limit is specified, below which the system will cease training. This value is used to prevent the ANN from continuing to train if the weights achieved are already producing very good results. Next, the initial η must be specified, as well as the final η value. MANN will continue to train until this final η value is reached (or the error limit is reached). At each η value a number of iterations are used to train the ANN, which must be specified to MANN. After the number of iterations is reached, η is reduced using a cooling schedule until it reaches the stopping η value.

The second set of parameters that must be specified is the structure of the ANN to be created. These are specified by providing three integer values to the MANN software, which represent the number of nodes in each of the three layers. Although this thesis is focusing on an ANN that can handle square tumours up to 3 cm in size, the MANN software was designed to be able to create any size of ANN. The number of input and output nodes will be dependent on what is used as the input and output to the ANN. The first experiment for the ANN will be to determine what the input and output should be. The second experimental component for the ANN will be to determine the number of interior nodes producing the best output.

Finally, the training mode of MANN must be specified. There are three training modes. The first training mode (0) is used to test the ANN that has been created, therefore the weights are loaded from a file and are not adjusted at all. In this mode the network simply generates output for the test data (recall mode). The second training mode (1) starts the weights at small random numbers and uses the training data to update the weights. The final training method (2) loads the weights from a previous training session and continues to modify them. A screen shot of the configuration dialog box for MANN is shown in Fig. 4.7.

4.3.2.3 MANN Output

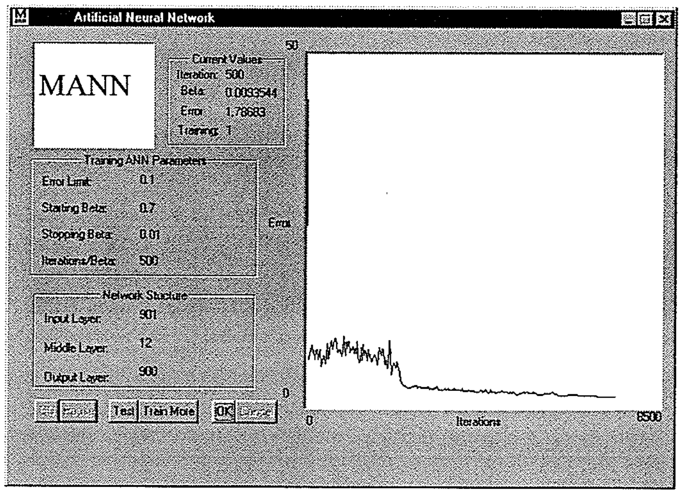
The most important output from the MANN program are the weights that are found with the BP algorithm. These weights are stored in a file, which can be loaded by BowANN as the representation of the trained ANN. The weights are not available until the ANN has completed training. Therefore, some real-time output was created to help assess the training process. A 2D graphics plot is displayed showing the current RMS error in the training process. This plot can be used for verification that the ANN is learning, as it should be a generally decreasing function. MANN also displays a number of current values in the system, such as the current η, the current iteration, and the numeric value of the current RMS error in the system, as shown in Fig. 4.8.


Configuration Input

Please enter the following values:

Error Limit:	0.1
Starting Beta:	0.75
Stopping Beta:	0.01
Iterations/Beta:	500
Input Layer:	901
Middle Layer:	12
Output Layer:	900
Training Mode:	1

Fig. 4.7 Configuration dialog box for MANN.


Artificial Neural Network

MANN

Current Values
Iteration: 500
Beta: 0.0033544
Error: 1.78683
Training: 1

Training ANN Parameters
Error Limit: 0.1
Starting Beta: 0.7
Stopping Beta: 0.01
Iterations/Beta: 500

Network Stucture
Input Layer: 901
Middle Layer: 12
Output Layer: 900

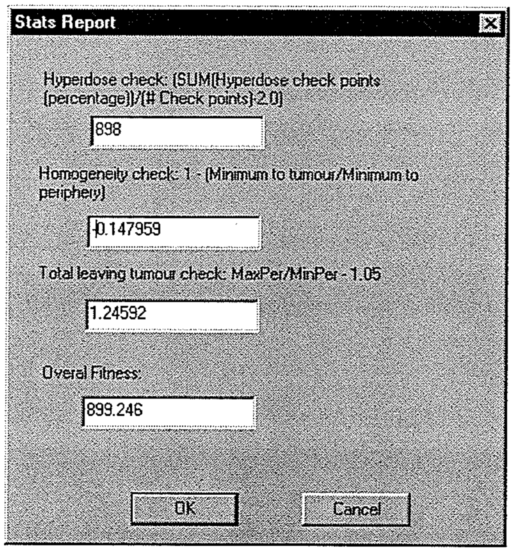
Fig. 4.8 MANN interface.

Once the ANN is created, it is tested. The quality of the ANN is characterized by the error in the output and this will be assessed at two stages. The first will be the difference in the optimal energy function calculated by the ANN compared to the energy function calculated by SA, and the second will be the final source positions as derived from the ANN energy function compared to the source positions of the SA. The source code for the MANN software can be found in Appendix D. It should be noted that some of the software code used was from Rao and Rao [RaRa95]. However, it was modified in the following ways. First, during an analysis of the software from [RaRa95] a memory leak problem was found and fixed. Second, a Windows wrapper was created around the [RaRa95] code so that it could be incorporated into Windows.

4.3.3 BowANN

The BowANN software provides a Windows based GUI to interface with the ANN trained in MANN for brachytherapy optimization. The program is able to optimize source positions within a tumour shape, which can either be loaded into the program as a *pgm* file or via the software interface. Next, the ratio (same as the ratio used in SAB and MANN) of the tumour can be specified (and normalized to the maximum value of 3 cm). BowANN is a Microsoft Windows based program for evaluating the effectiveness of the trained ANN and the optimal source positions that it yields.

BowANN uses the ANN trained in MANN to perform optimization. The user has a standard Windows 95/98/NT GUI with which to manipulate the environment. The user has the ability to add additional sources (other than the ANN generated source positions) using the mouse or by entering the actual physical coordinates into a dialog box. Energy functions calculated based on the sources in the environment can be displayed using OpenGL and 3D computer graphics. The energy functions can be rotated in order to be assessed from any angle. Finally, the user can also view statistics on the current source configuration in a dialog box that reports values associated with the cost function (Section 4.3.1.5) from SAB as shown in Fig. 4.9.


Stats Report

Hyperdose check: (SUM(Hyperdose check points (percentage))/(# Check points)-2.0)

898

Homogeneity check: 1 - (Minimum to tumour/Minimum to periphery)

-0.147959

Total leaving tumour check: MaxPer/MinPer - 1.05

1.24592

Overal Fitness:

899.246

Fig. 4.9 BowANN statistics report dialog box.

The BowANN software is typically used as follows. A tumour is loaded from a pgm file. The user then instructs the software to find the optimal source positions for the tumour, and displays them as probability distributions. The peak of the distribution represents the best location for the source, as shown in Fig. 4.10.

The user can then simulate a brachytherapy insertion procedure by placing a source at a point in the tumour, which would represent an inserted source, and then instruct the software to re-calculate the optimal positions of additional sources (given that a source has been inserted in the tumour). BowANN will then generate new output based on the new input as shown in Fig. 4.11

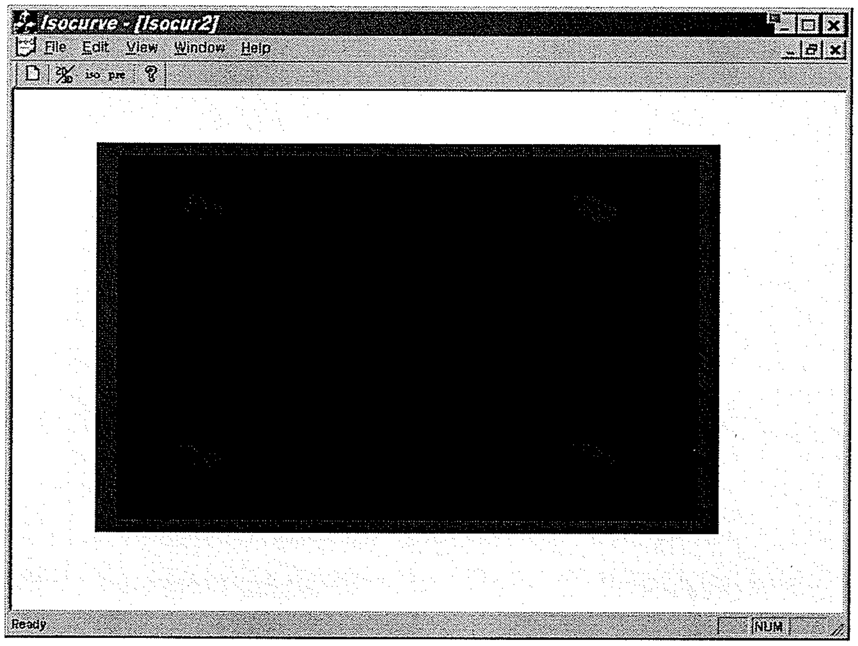


Fig. 4.10 Predicted source positions in BowANN.

In this manner a brachytherapy insertion is simulated, and BowANN is used to view what the ANN would generate as the desired positions for all sources yet to be inserted. At anytime, the user can also plot the current dose distribution in the tumour with a simple command to the BowANN software. This dose is normalized to the minimum dose on the periphery of the tumour which is indicated as red on the display. The output from the isodose plot can be viewed in 2D as shown in Fig. 4.12, or in 3D as shown in Fig. 4.13. The source code for the BowANN software can be found in Appendix E.

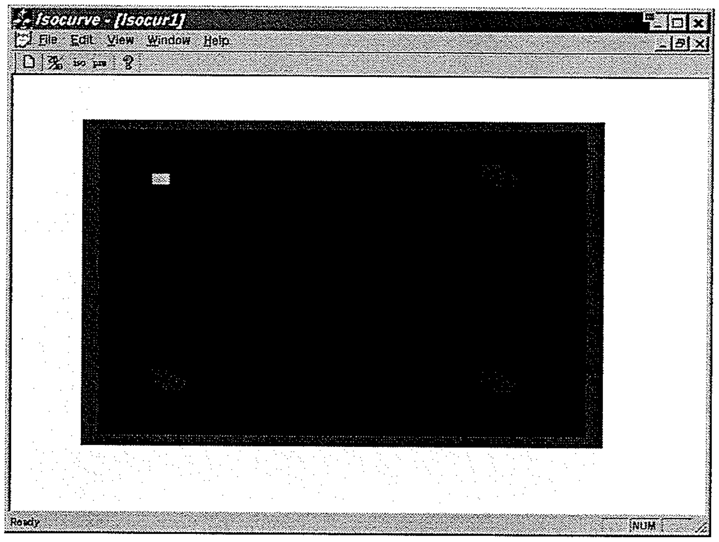


Fig. 4.11 Updated predicted source positions in BowANN.

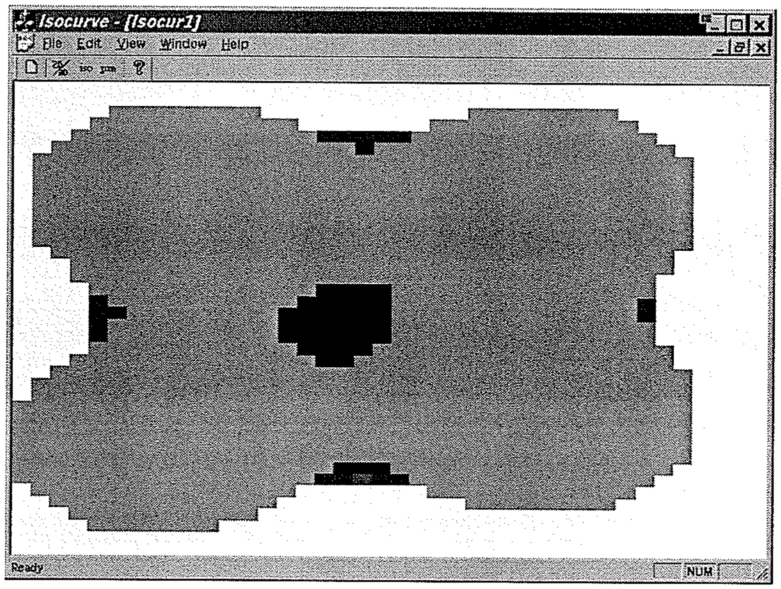


Fig. 4.12 2D 125% isodose plot in BowANN.

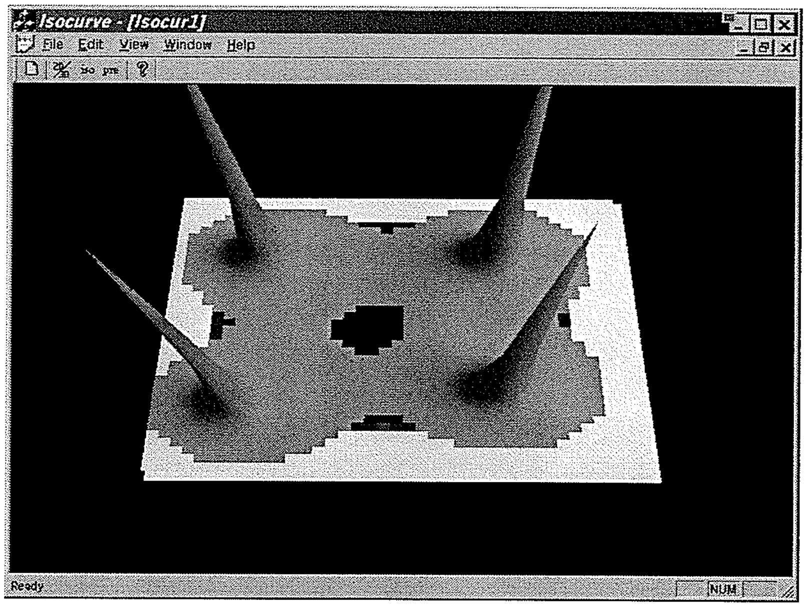


Fig. 4.13 3D isodose plot in BowANN.

4.4 Chapter Summary

This Chapter identified the breakdown of the overall brachytherapy optimization problem into three components, SAB, MANN, and BowANN. The available hardware technology was mapped to each of the components based on their requirements. All of the inputs and outputs of the software programs and parameters used to control them were identified. The following Chapter designs and conducts experiments that answer the three main questions of this thesis. These questions are: What should be used as input to an ANN for brachytherapy optimization?; What should be used as output?; and what is the best form of a BP ANN to optimize brachytherapy treatments?

CHAPTER V

EXPERIMENTAL RESULTS AND DISCUSSION

"Nothing stops the man who desires to achieve. Every obstacle is simply a course to develop his achievement muscle. It's a strengthening of his powers of accomplishment."
- Eric Butterworth, inspirational speaker

5.1 Introduction

The purpose of performing experiments is to verify that the concepts as well as the methods used to solve a problem are correct. For this thesis we wish to prove that ANNs can be used to optimize brachytherapy treatments. In order to prove this we must ensure that the software design and implementation work and meet the requirements identified. For the SA aspect of the thesis we want to ensure that SAB meets the requirements of Section 3.2 and is finding correct training data for the ANN. For the ANN aspect of the thesis we want to ensure that it meets the requirements identified in Sections 3.3 and 3.4. Experiments will also be used to answer the three main questions of thesis in regards to ANNs: (i) Can an ANN be used for brachytherapy optimization? (ii) What should be used as input and output to the ANN? and (iii) What is the best structure of an ANN for brachytherapy optimization? The following sections will describe the design of the experiments used and present results.

5.2 Experimental Design for SA

The results from the SAB software are used to train an ANN for brachytherapy optimization. In order to prove that an ANN can be used for brachytherapy optimization, it must be trained using realistic training data. Therefore, it is important to ensure that the SAB software generates correct output. In order to verify this, a number of different tumour sizes will be used as input to the SAB software and the output is evaluated. As this thesis is limited to square tumours up to 3 cm in size we will experiment with inputs ranging from 0.5 cm to 3.0 cm in 0.5 cm increments. It should be mentioned that square tumours are more difficult to optimize than the more realistic elliptical tumours are the corners represent singularity. The input to the SAB software is the tumour shapes (to optimize) as well as a configuration file (to control the flow of the SAB software). The output for each of the tumour sizes is collected and evaluated to ensure that it is correct.

5.3 Experimental Results and Discussion for SA

The following tumour sizes are used as input to the SAB software: 0.5 cm, 1.0cm, 1.5 cm, 2.0 cm, 2.5 cm and 3.0 cm. The following sections will cover the results for these sizes. Additional sizes were also used, however a detailed discussion will only be conducted on the aforementioned sizes. To view the results from the additional experiments, consult Appendix A. During initial testing of the SAB software it was determined that a ratio of 3 seemed to work the best. This means that 3 pixels are used to represent 1 mm. Since this standard has been identified it will be used for all input sizes. It was also found that a cooling schedule that uses approximately 10,000 iterations at each temperature works best, when combined with the cooling profile shown in Fig. 2.5.

5.3.1 Square Tumour 0.5 cm in Width

The 0.5 cm tumour shown in Fig. 5.1 was used as input to the SAB software. The configuration file shown in Fig. 5.2 was used to control the flow of the SAB software as discussed in Section 4.3.1.2. This trial produced a cost iteration plot as shown in Fig. 5.3. The cooling profile shown in Fig. 5.4 was utilized. The output from the SAB software is as shown in Fig. 5.5. The hyperdose sleeve for the solution shown in Fig. 5.5, is shown in Fig. 5.6. Since the cost function for SAB is essentially comprised of two factors, the homogeneity of the dose in a tumour and the size of the hyperdose sleeves, it is easy to predict the output for a small tumour (0.5 cm). The results shown in Fig. 5.5 are exactly as expected. Since the tumour size is smaller than the maximum hyperdose sleeve size, a single source could be placed anywhere in the tumour and meet that requirement in the SA cost function. However, because of the second component of the SA cost function a homogeneous dose in the tumour is more desirable. As the SAB software attempts to find the best solution with the minimum number of sources possible, a single source at the center is exactly as expected. The output from the SAB algorithm for a tumour size of 0.5 cm is therefore shown to be correct. The final cost function value for this source distribution was 3.035926.

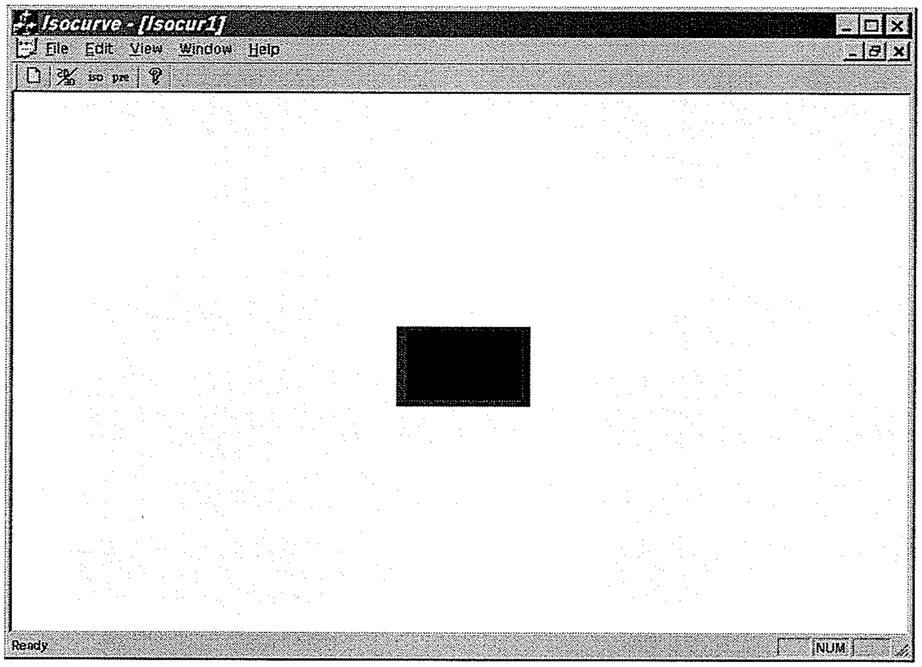
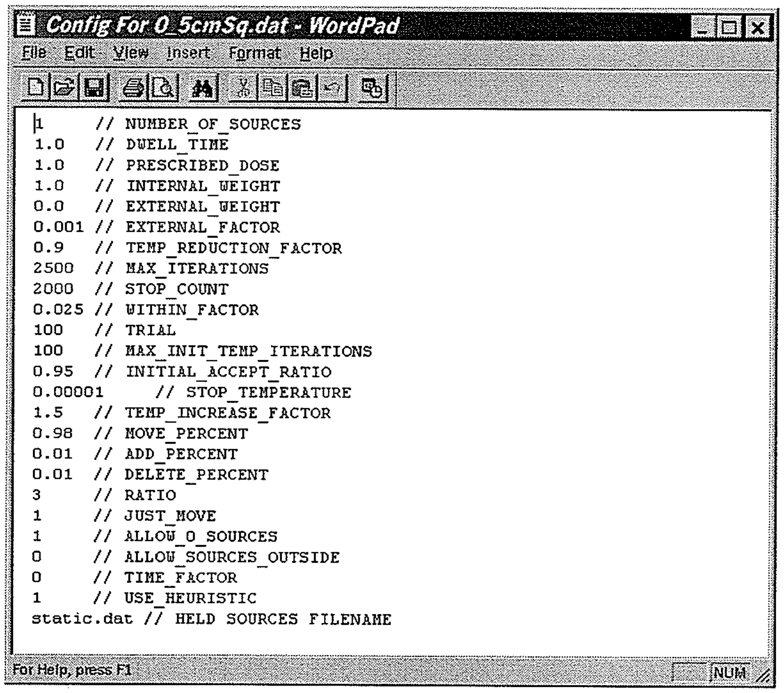


Fig. 5.1 0.5 cm tumour to be optimized with SAB.


Config For 0_5cmSq.dat - WordPad
File Edit View Insert Format Help
For Help, press F1
NUM

```
1      // NUMBER_OF_SOURCES
1.0    // DWELL_TIME
1.0    // PRESCRIBED_DOSE
1.0    // INTERNAL_WEIGHT
0.0    // EXTERNAL_WEIGHT
0.001  // EXTERNAL_FACTOR
0.9    // TEMP_REDUCTION_FACTOR
2500   // MAX_ITERATIONS
2000   // STOP_COUNT
0.025  // WITHIN_FACTOR
100    // TRIAL
100    // MAX_INIT_TEMP_ITERATIONS
0.95   // INITIAL_ACCEPT_RATIO
0.00001      // STOP_TEMPERATURE
1.5    // TEMP_INCREASE_FACTOR
0.98   // MOVE_PERCENT
0.01   // ADD_PERCENT
0.01   // DELETE_PERCENT
3      // RATIO
1      // JUST_MOVE
1      // ALLOW_0_SOURCES
0      // ALLOW_SOURCES_OUTSIDE
0      // TIME_FACTOR
1      // USE_HEURISTIC
static.dat // HELD SOURCES FILENAME
```

Fig. 5.2 Configuration file for 0.5 cm tumour.

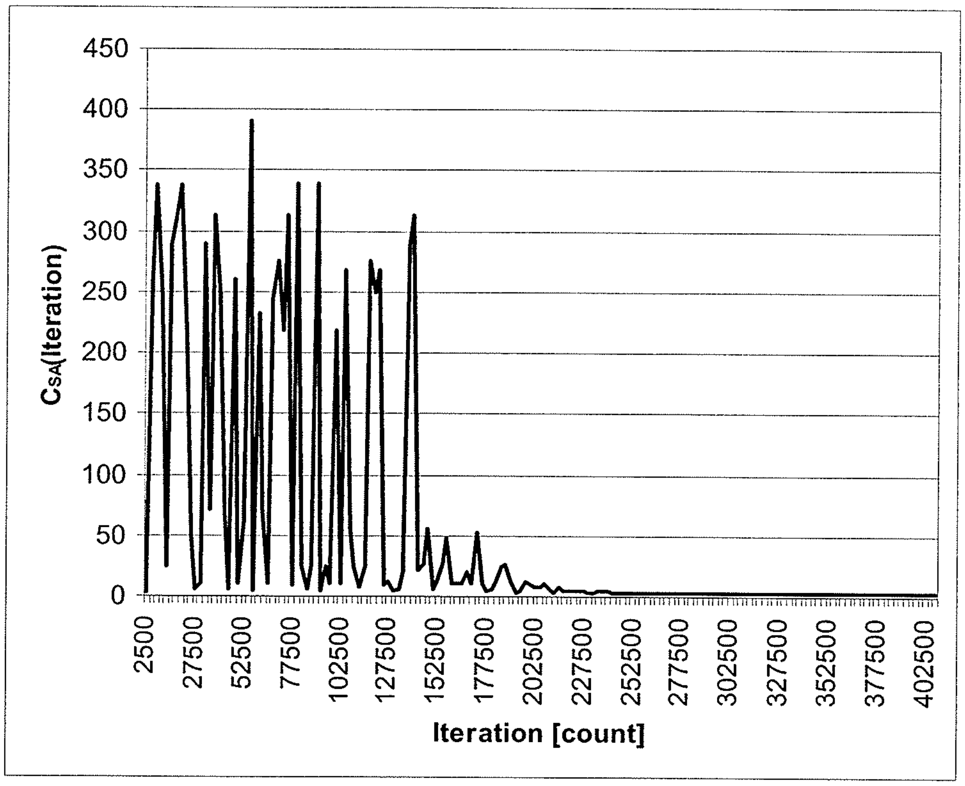


Fig. 5.3 Cost plot for 0.5 cm tumour.

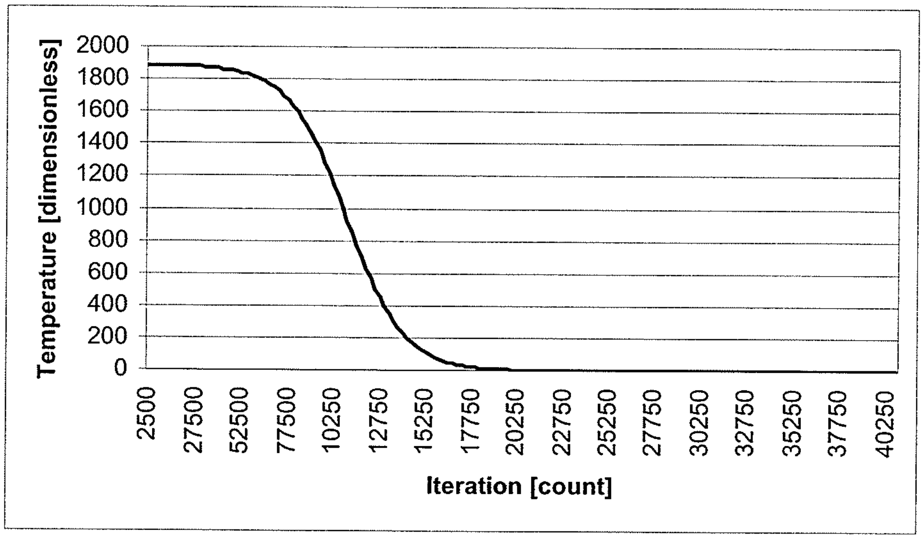
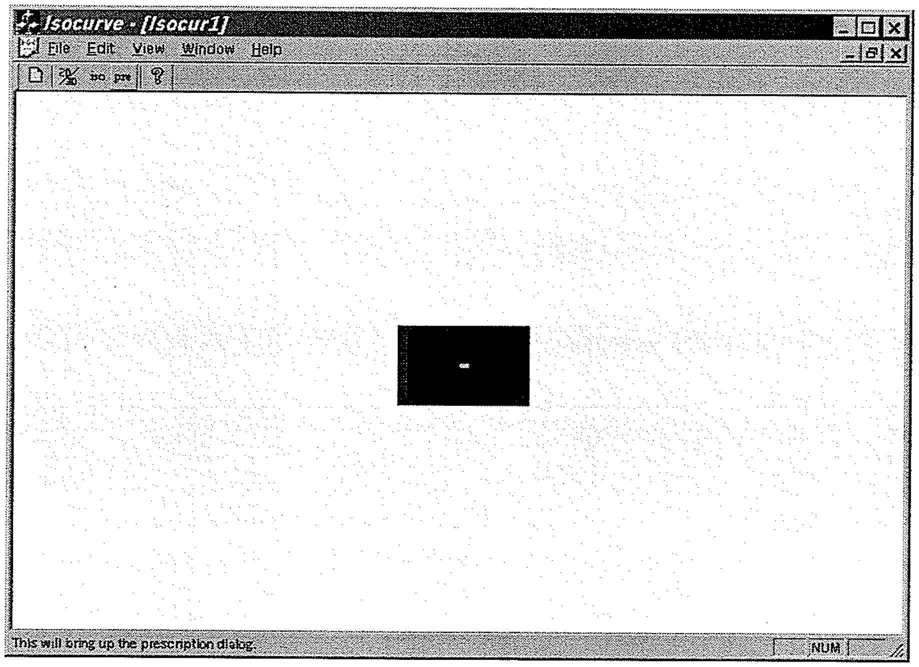


Fig. 5.4 Cooling profile for 0.5 cm tumour.



This will bring up the prescription dialog.

Fig. 5.5 Optimized source positions for 0.5 cm tumour.

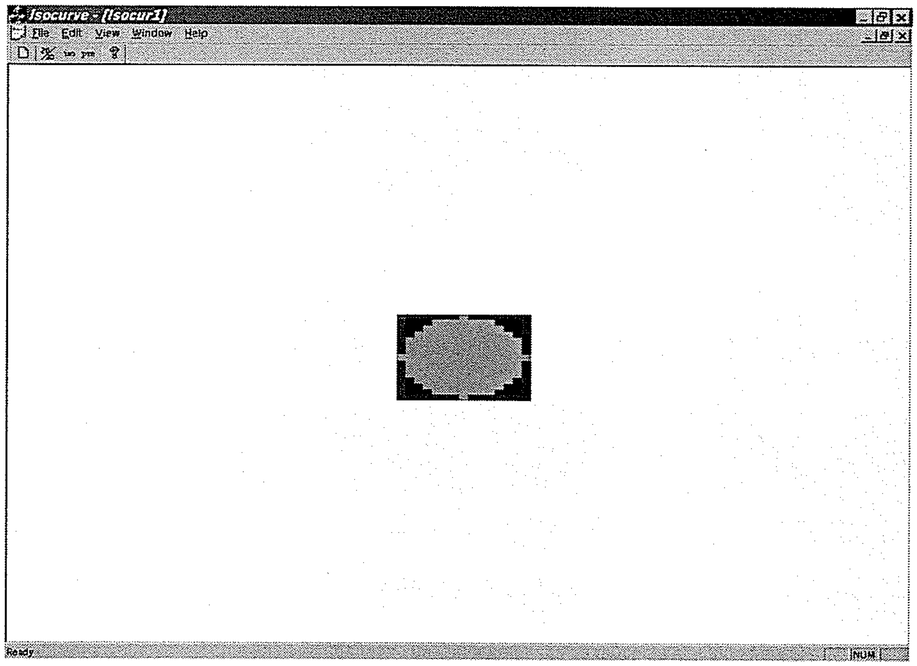


Fig. 5.6 Hyperdose sleeve for 0.5 cm solution.

5.3.2 Square Tumour 1.0 cm in Width

The 1.0 cm tumour shown in Fig. 5.7 was used as input to the SAB software. The configuration file shown in Fig. 5.8 was used to control the flow of the SAB software. This trial produced a cost iteration plot as shown in Fig. 5.9. The number of iterations for the optimization of the 1.0 cm tumour is considerable less then for the 0.5 cm tumour as the SAB software finds a perfect solution. With a 1.0 cm tumour the cost function can be satisfied exactly with one source at the center. If the SAB software ever reaches a point in which 2000 iterations (as specified in the configuration file of Fig. 5.8) in a row occur without a change in the cost occurs it will stop and return the result as the answer. A cooling profile shown in Fig. 5.10 was utilized Which has less iterations than the cooling profile of Fig. 5.4 because the stopping conditions of 2000 iterations without change was met. The output from the SAB software is as shown in Fig. 5.11. The hyperdose sleeve for the solution shown in Fig. 5.11, is shown in Fig. 5.12. The optimal source distribution for a 1.0 cm square is easy to predict as it should contain only a single source, using a logic similar to that in the previous section. A final cost value of 1.245919 was obtained by SAB for the 1.0 cm square tumour.

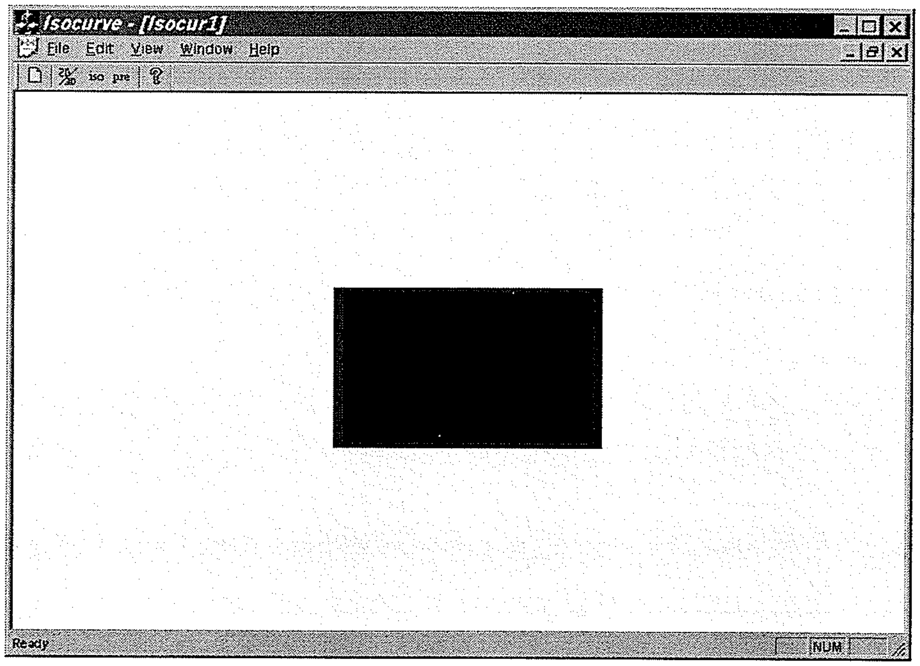
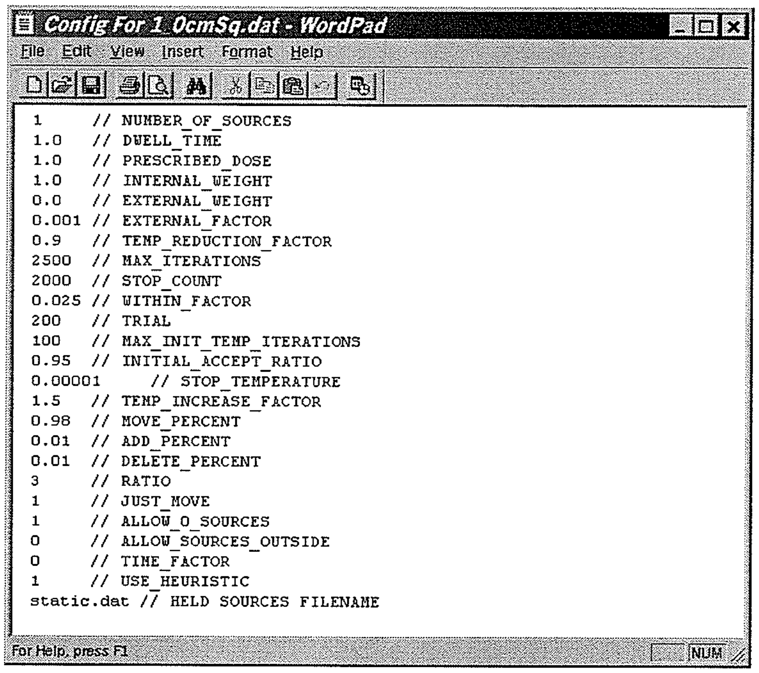


Fig. 5.7 1.0 cm tumour to be optimized with SAB.


Config For 1 OcmSq.dat - WordPad
File Edit View Insert Format Help
For Help, press F1
NUM

```
1      // NUMBER_OF_SOURCES
1.0    // DWELL_TIME
1.0    // PRESCRIBED_DOSE
1.0    // INTERNAL_WEIGHT
0.0    // EXTERNAL_WEIGHT
0.001 // EXTERNAL_FACTOR
0.9    // TEMP_REDUCTION_FACTOR
2500  // MAX_ITERATIONS
2000  // STOP_COUNT
0.025 // WITHIN_FACTOR
200    // TRIAL
100    // MAX_INIT_TEMP_ITERATIONS
0.95  // INITIAL_ACCEPT_RATIO
0.00001      // STOP_TEMPERATURE
1.5    // TEMP_INCREASE_FACTOR
0.98  // MOVE_PERCENT
0.01  // ADD_PERCENT
0.01  // DELETE_PERCENT
3      // RATIO
1      // JUST_MOVE
1      // ALLOW_0_SOURCES
0      // ALLOW_SOURCES_OUTSIDE
0      // TIME_FACTOR
1      // USE_HEURISTIC
static.dat // HELD SOURCES FILENAME
```

Fig. 5.8 Configuration file for 1.0 cm tumour.

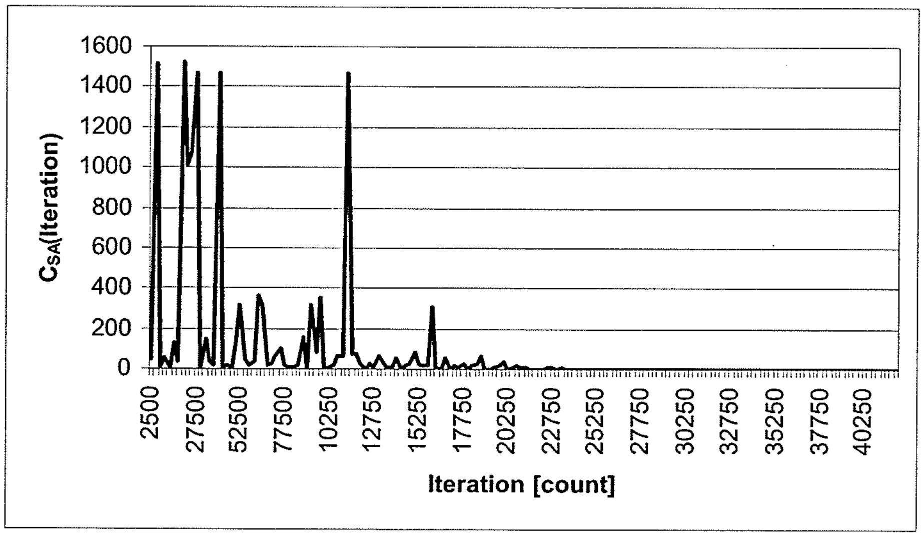


Fig. 5.9 Cost plot for 1.0 cm tumour.

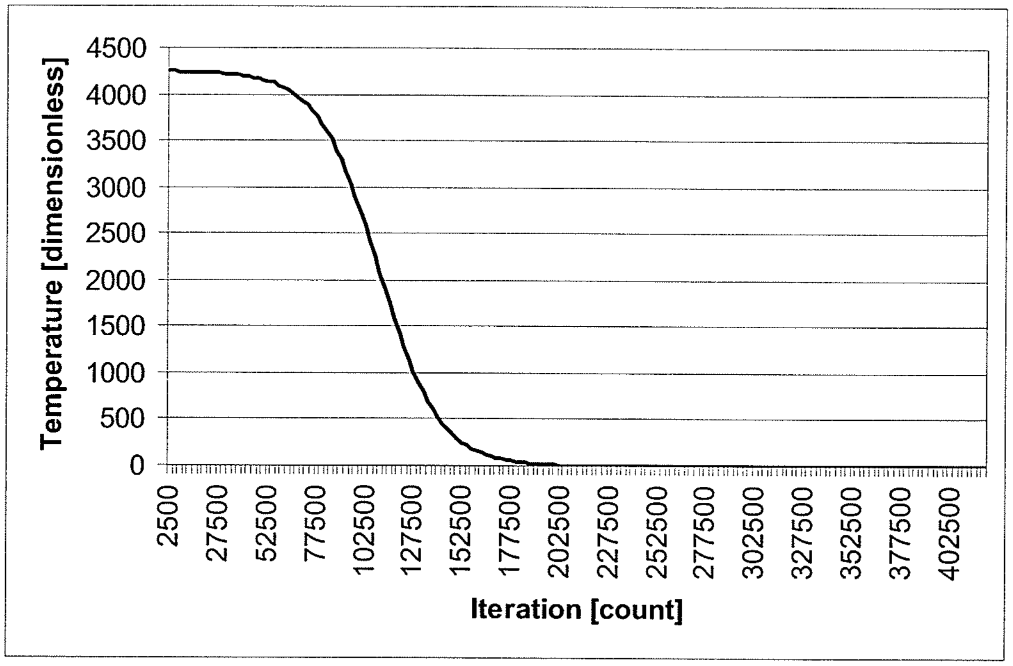


Fig. 5.10 Cooling profile for 1.0 cm tumour.

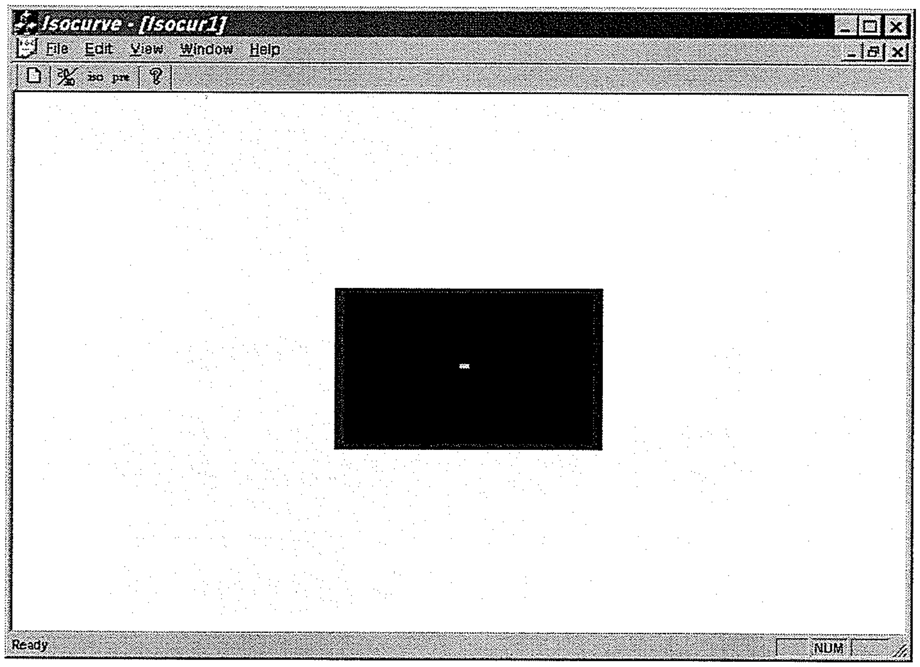


Fig. 5.11 Optimized source positions for 1.0 cm tumour.

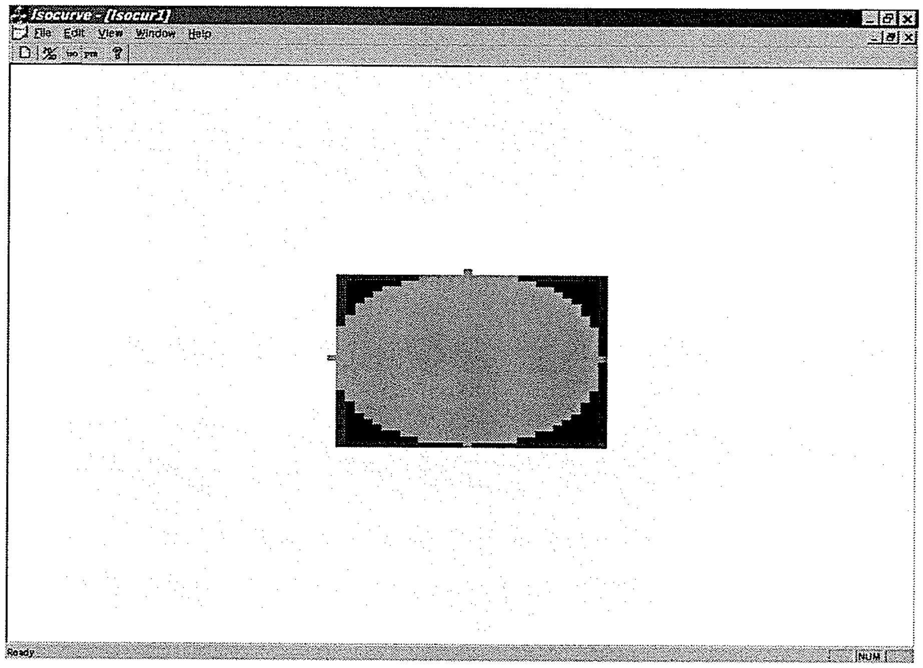


Fig. 5.12 Hyperdose sleeve for 1.0 cm solution.

5.3.3 Square Tumour 1.5 cm in Width

The 1.5 cm tumour shown in Fig. 5.13 was used as input to the SAB software. The configuration file shown in Fig. 5.14 was used to control the flow of the SAB software. This trial produced a cost iteration plot as shown in Fig. 5.15. The plot shown in Fig. 5.15 is unique compared to the previous plots presented in this thesis as there are four distinct sections with increased cost. Each of these pulses correspond to an additional source being added to the simulation. Similarly the plot shown in Fig. 5.16 has four distinct cooling phases. Sources are added in a controlled fashion in order to minimize the number of sources (or needles) used. Thus the number of sources is not included in the cost function but it is minimized by adding sources in a controlled fashion. The actual progression of point sources being added is shown for the 3.0 cm tumour in an upcoming section (Section 5.3.6). A cooling profile shown in Fig. 5.16 was utilized. The output from the SAB software is as shown in Fig. 5.17. The hyperdose sleeve for the solution shown in Fig. 5.17, is shown in Fig. 5.18. These hyperdose sleeves only have a cross sectional size of 6.3333 mm, well within the 1.0 cm limit. The output obtained is what is expected, a homogeneous distribution of sources, hyperdose sleeves smaller than 1.0 cm and a final cost function value of 1.613671.

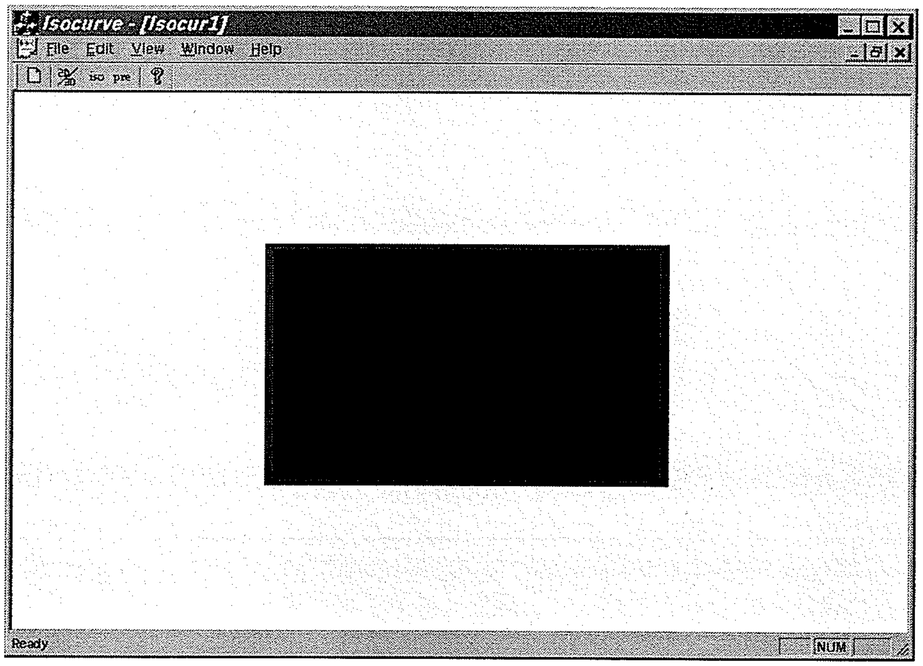
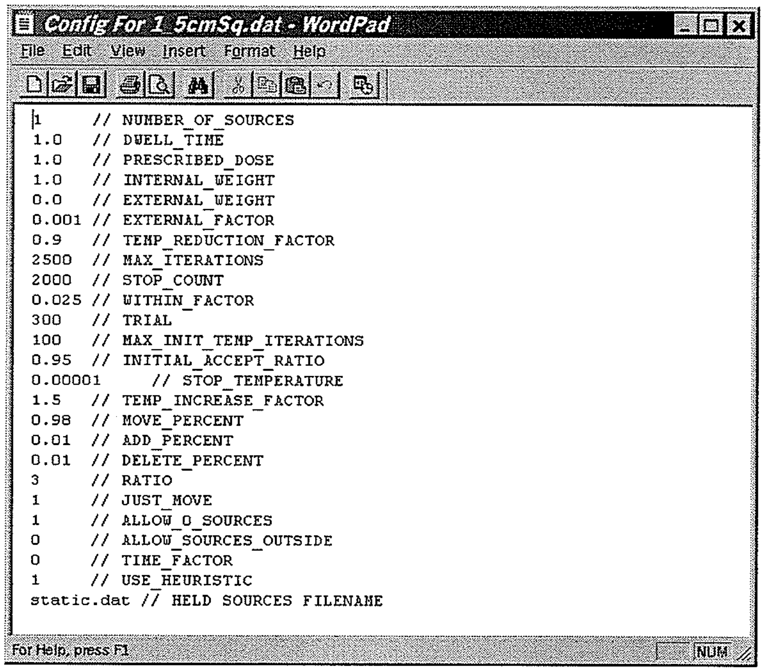


Fig. 5.13 1.5 cm tumour to be optimized with SAB.


Config For 1_5cmSq.dat - WordPad
File Edit View Insert Format Help
For Help, press F1
NUM

```
1      // NUMBER_OF_SOURCES
1.0    // DWELL_TIME
1.0    // PRESCRIBED_DOSE
1.0    // INTERNAL_WEIGHT
0.0    // EXTERNAL_WEIGHT
0.001 // EXTERNAL_FACTOR
0.9    // TEMP_REDUCTION_FACTOR
2500  // MAX_ITERATIONS
2000  // STOP_COUNT
0.025 // WITHIN_FACTOR
300    // TRIAL
100    // MAX_INIT_TEMP_ITERATIONS
0.95  // INITIAL_ACCEPT_RATIO
0.00001      // STOP_TEMPERATURE
1.5    // TEMP_INCREASE_FACTOR
0.98  // MOVE_PERCENT
0.01  // ADD_PERCENT
0.01  // DELETE_PERCENT
3      // RATIO
1      // JUST_MOVE
1      // ALLOW_0_SOURCES
0      // ALLOW_SOURCES_OUTSIDE
0      // TIME_FACTOR
1      // USE_HEURISTIC
static.dat // HELD SOURCES FILENAME
```

Fig. 5.14 Configuration file for 1.5 cm tumour.

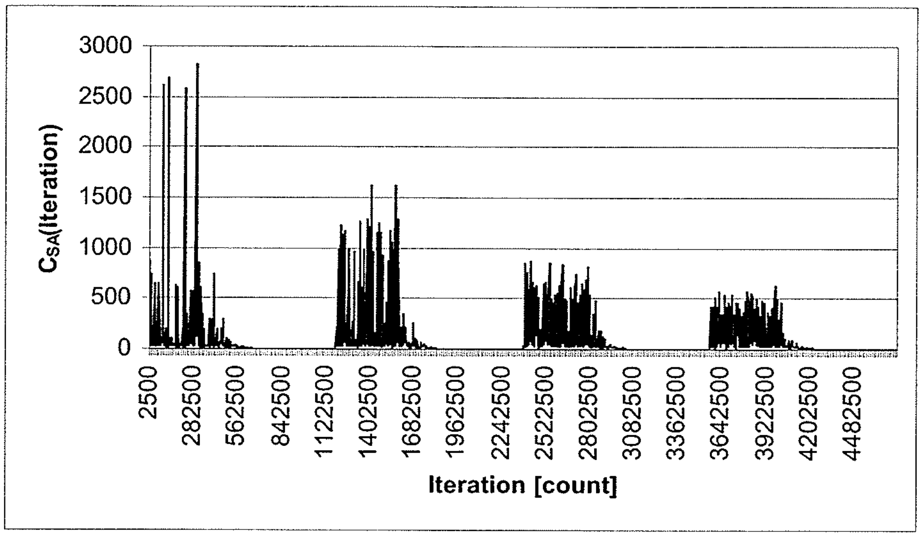


Fig. 5.15 Cost plot for 1.5 cm tumour.

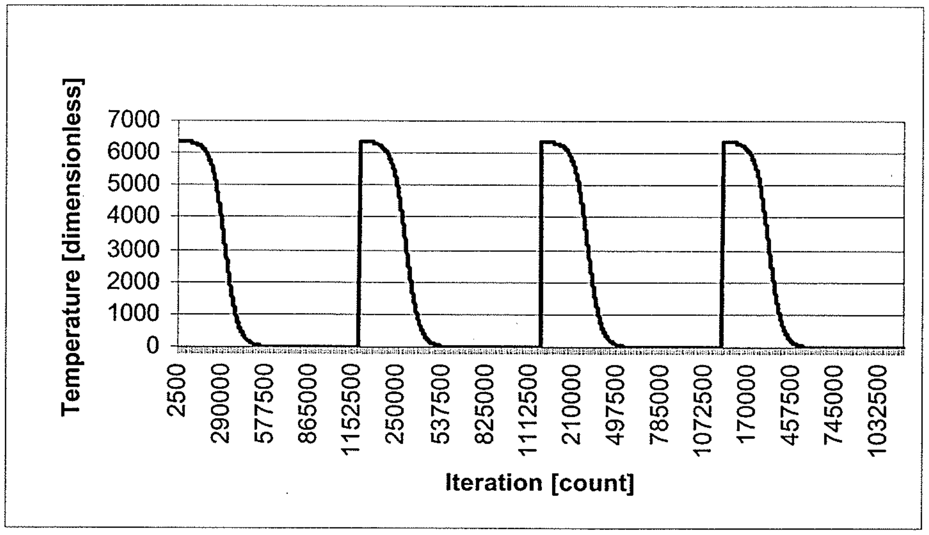


Fig. 5.16 Cooling profile for 1.5 cm tumour.

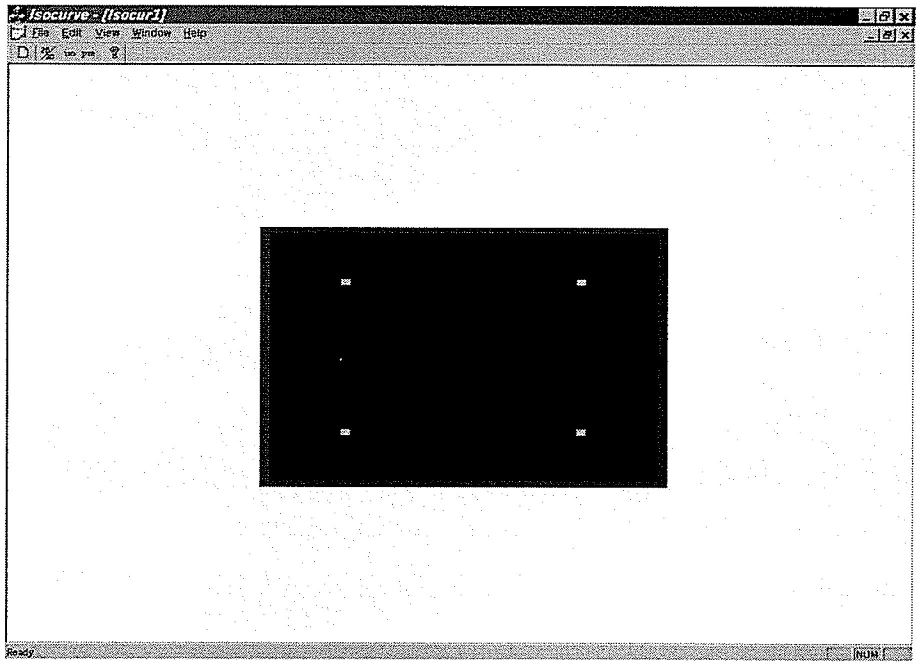


Fig. 5.17 Optimized source positions for 1.5 cm tumour.

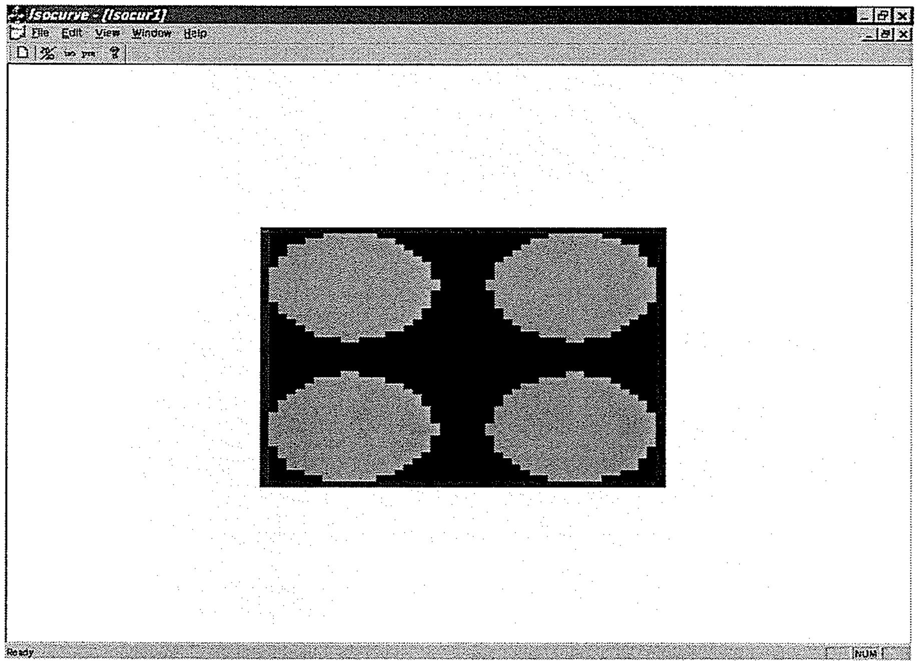


Fig. 5.18 Hyperdose sleeve for 1.5 cm solution.

5.3.4 Square Tumour 2.0 cm in Width

The 2.0 cm tumour shown in Fig. 5.19 was used as input to the SAB software. The configuration file shown in Fig. 5.20 was used to control the flow of the SAB software. This trial produced a cost iteration plot as shown in Fig. 5.21. Similar to the discussion in the previous section, four distinct phases are seen in 5.21 corresponding to each of the four sources in the final solution being added. To see the progression of a solution completely see Section 5.3.6. A cooling profile shown in Fig. 5.22 was utilized. The output from the SAB software is as shown in Fig. 5.23. The hyperdose sleeve for the solution shown in Fig. 5.11, is shown in Fig. 5.12. Unfortunately the results for a 2.0 cm square tumour leave a little bit to be desired. However, upon careful inspection, and further experimentation, it was found that the results found by SAB seem to be the optimal source positions even though the resulting hyperdose sleeves are too big. There are a couple of possible explanations for why this occurs. The first is that perhaps there is no such thing as a perfect placement of equal strength sources in a 2.0 cm tumour. The second possible explanation is that perhaps if sources were able to be placed external to the tumour a better hyperdose sleeve could be achieved. When the SAB software continued adding source, it reached the maximum number of sources allowed in a simulation (currently 15) and was unable to find a suitable answer. The solutions with more than four sources had cost functions with cost values higher than the final cost function value for four sources. The best result found is the one presented here. This imperfect result shows the complexity of trying to optimize brachytherapy implants. The final cost function value for four sources in a 2.0 cm tumour is 1.759948. It should be noted that a plot of the 218% dose yields an isodose line that corresponds to the maximum acceptable size for the hyperdose sleeves, so this solution is approximately 18% off what we require.

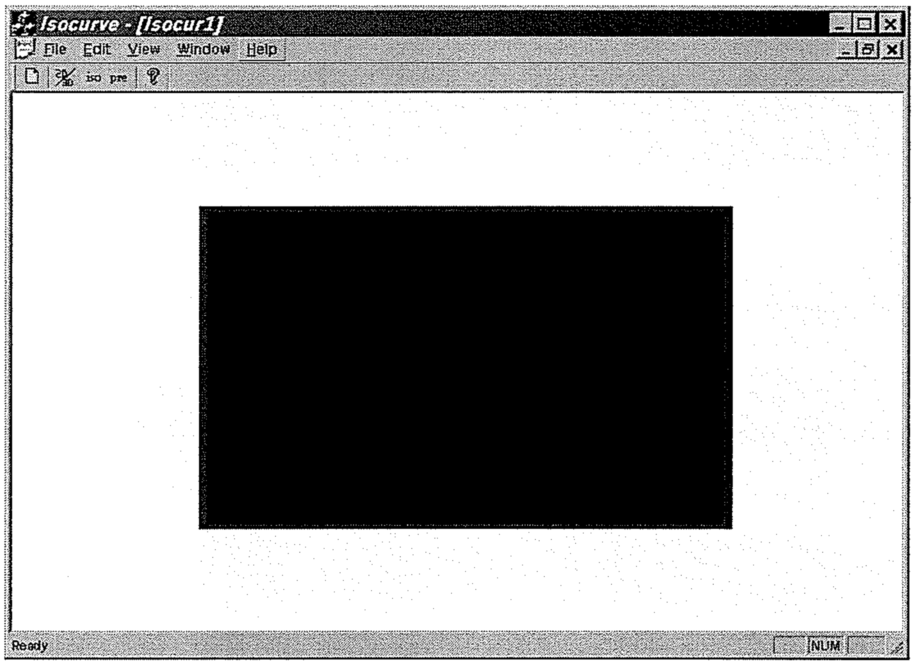
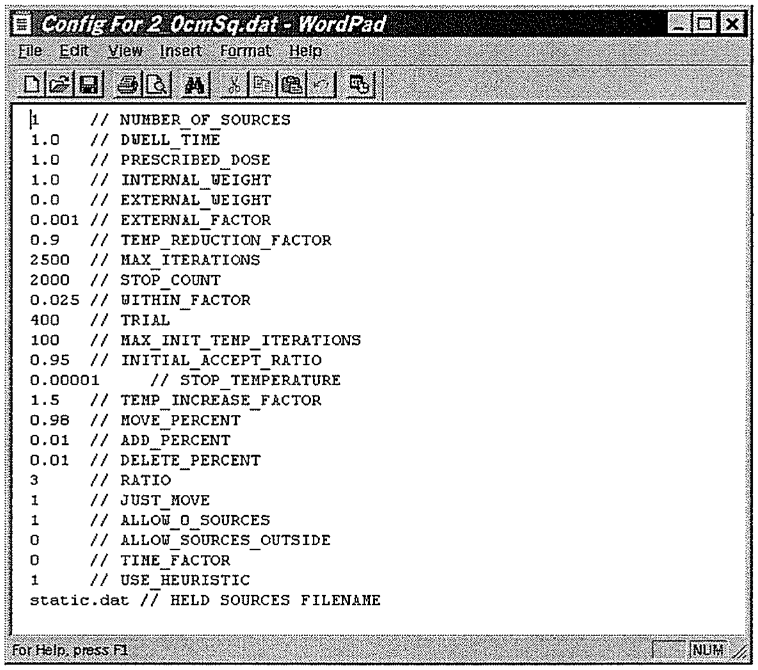


Fig. 5.19 2.0 cm tumour to be optimized with SAB.


Config For 2_0cmSq.dat - WordPad
File Edit View Insert Format Help
For Help, press F1
NUM

```
1      // NUMBER_OF_SOURCES
1.0    // DWELL_TIME
1.0    // PRESCRIBED_DOSE
1.0    // INTERNAL_WEIGHT
0.0    // EXTERNAL_WEIGHT
0.001 // EXTERNAL_FACTOR
0.9    // TEMP_REDUCTION_FACTOR
2500  // MAX_ITERATIONS
2000  // STOP_COUNT
0.025 // WITHIN_FACTOR
400    // TRIAL
100    // MAX_INIT_TEMP_ITERATIONS
0.95  // INITIAL_ACCEPT_RATIO
0.00001      // STOP_TEMPERATURE
1.5    // TEMP_INCREASE_FACTOR
0.98  // MOVE_PERCENT
0.01  // ADD_PERCENT
0.01  // DELETE_PERCENT
3      // RATIO
1      // JUST_MOVE
1      // ALLOW_0_SOURCES
0      // ALLOW_SOURCES_OUTSIDE
0      // TIME_FACTOR
1      // USE_HEURISTIC
static.dat // HELD SOURCES FILENAME
```

Fig. 5.20 Configuration file for 2.0 cm tumour.

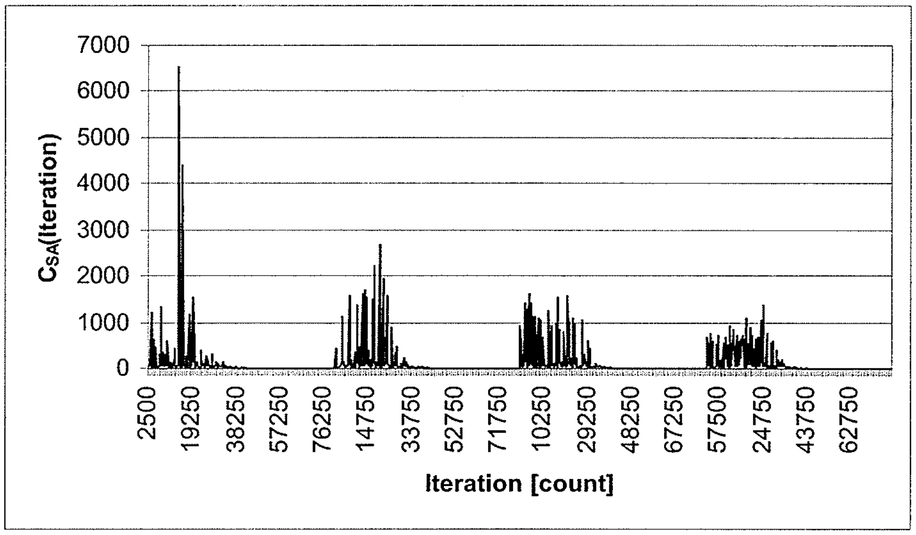


Fig. 5.21 Cost plot for 2.0 cm tumour.

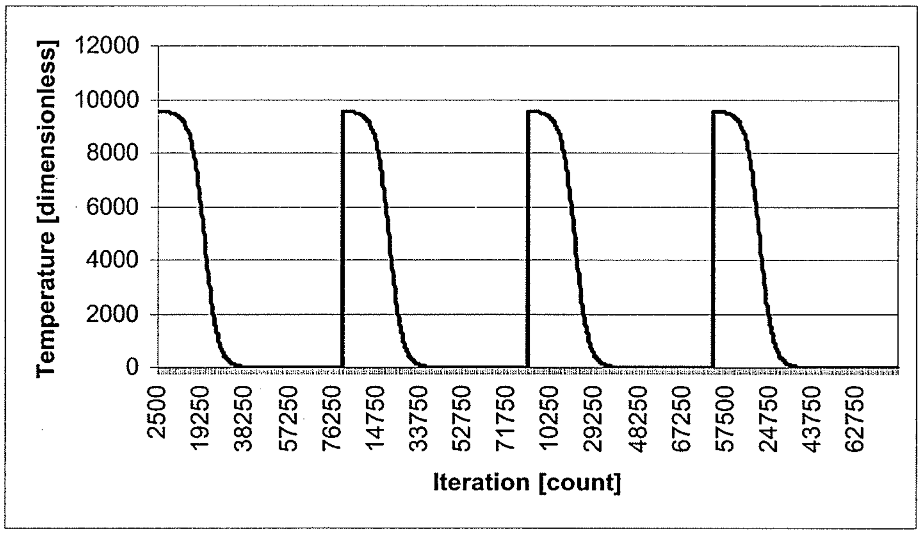


Fig. 5.22 Cooling profile for 2.0 cm tumour.

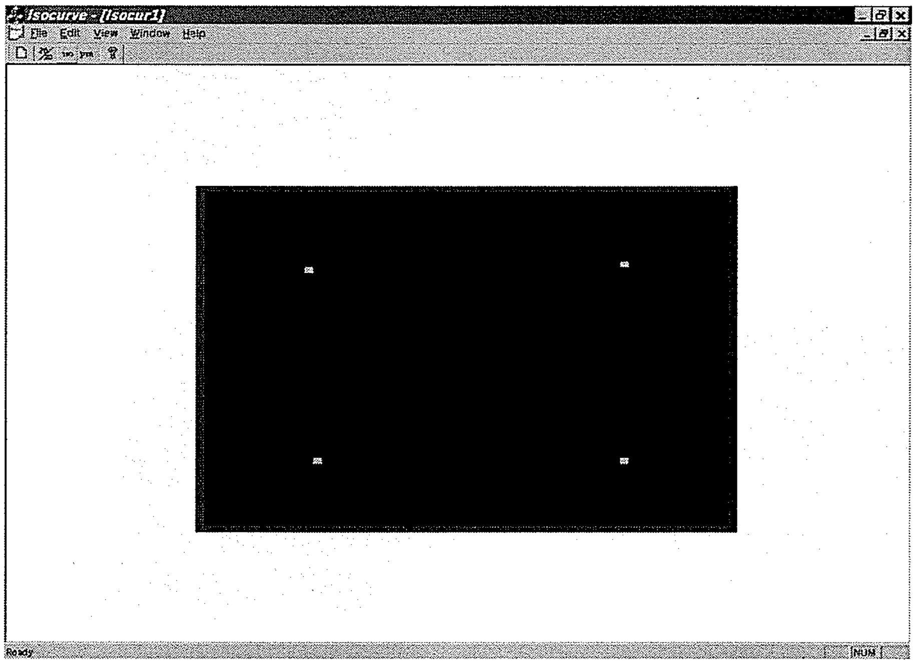


Fig. 5.23 Optimized source positions for 2.0 cm tumour.

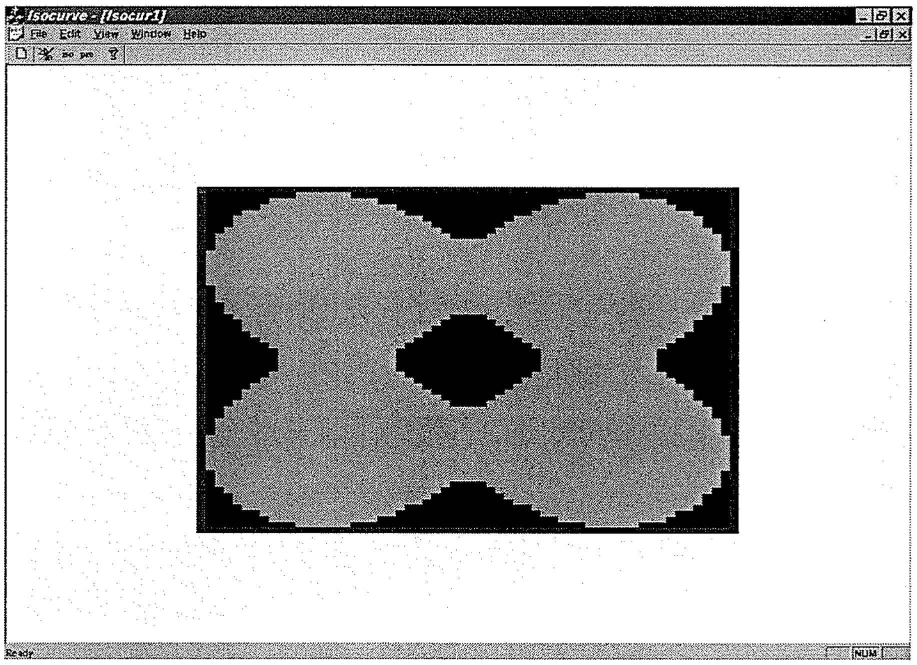


Fig. 5.24 Hyperdose sleeve for 2.0 cm solution.

5.3.5 Square Tumour 2.5 cm in Width

The tumour shown in Fig. 5.25 was used as input to the SAB software. The configuration file shown in Fig. 5.26 was used to control the flow of the SAB software. This trial produced a cost iteration plot as shown in Fig. 5.27. Similar to the discussion in the previous section, four distinct phases are seen in 5.27 corresponding to each of the four sources in the final solution being added. To see the progression of a solution completely see Section 5.3.6. A cooling profile shown in Fig. 5.28 was utilized. The output from the SAB software is as shown in Fig. 5.29. The hyperdose sleeve for the solution in Fig. 5.29, is shown in Fig. 5.30. The hyperdose sleeves in Fig. 5.30 have exactly a 1.0 cm diameter which is the maximum acceptable size according to our cost function. The final cost function value is 1.098585.

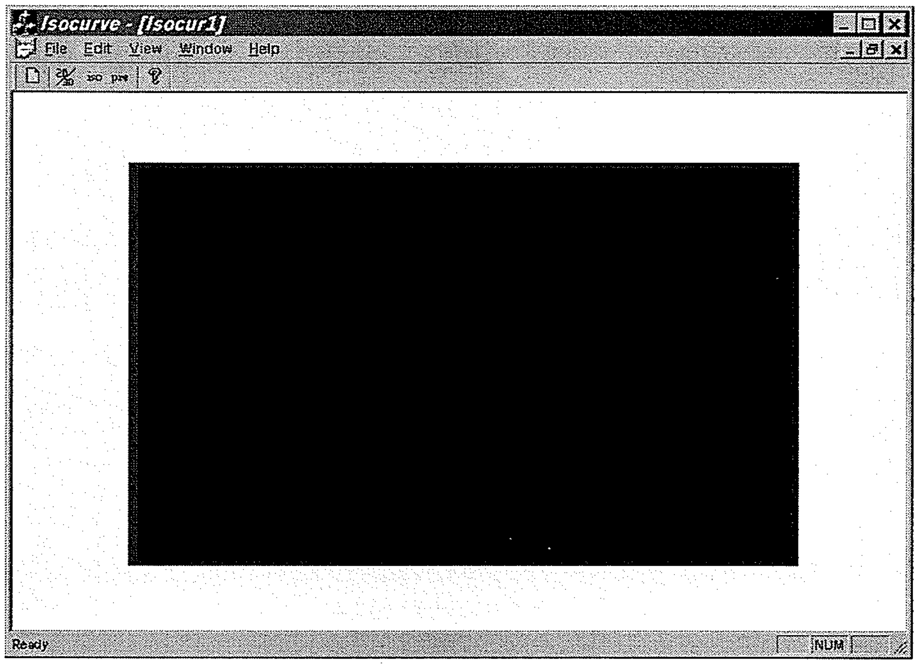
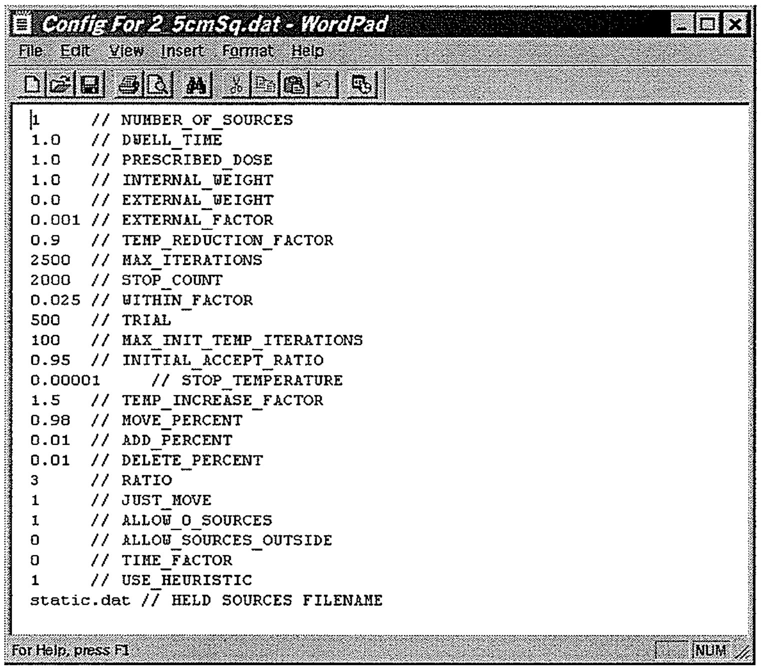


Fig. 5.25 2.5 cm tumour to be optimized with SAB.


Config For 2_5cmSq.dat - WordPad
File Edit View Insert Format Help
For Help, press F1
NUM

```
1      // NUMBER_OF_SOURCES
1.0    // DWELL_TIME
1.0    // PRESCRIBED_DOSE
1.0    // INTERNAL_WEIGHT
0.0    // EXTERNAL_WEIGHT
0.001 // EXTERNAL_FACTOR
0.9    // TEMP_REDUCTION_FACTOR
2500  // MAX_ITERATIONS
2000  // STOP_COUNT
0.025 // WITHIN_FACTOR
500    // TRIAL
100    // MAX_INIT_TEMP_ITERATIONS
0.95  // INITIAL_ACCEPT_RATIO
0.00001     // STOP_TEMPERATURE
1.5    // TEMP_INCREASE_FACTOR
0.98  // MOVE_PERCENT
0.01  // ADD_PERCENT
0.01  // DELETE_PERCENT
3      // RATIO
1      // JUST_MOVE
1      // ALLOW_0_SOURCES
0      // ALLOW_SOURCES_OUTSIDE
0      // TIME_FACTOR
1      // USE_HEURISTIC
static.dat // HELD SOURCES FILENAME
```

Fig. 5.26 Configuration file for 2.5 cm tumour.

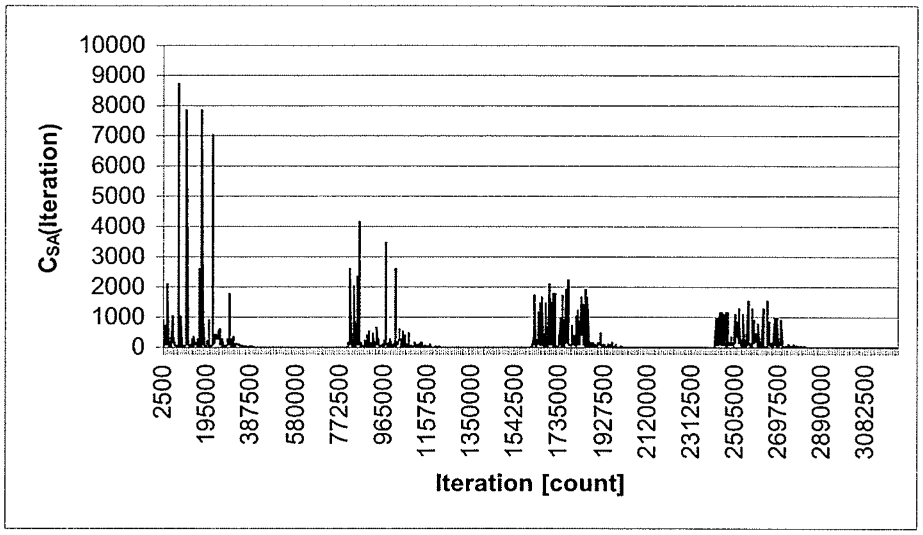


Fig. 5.27 Cost plot for 2.5 cm tumour.

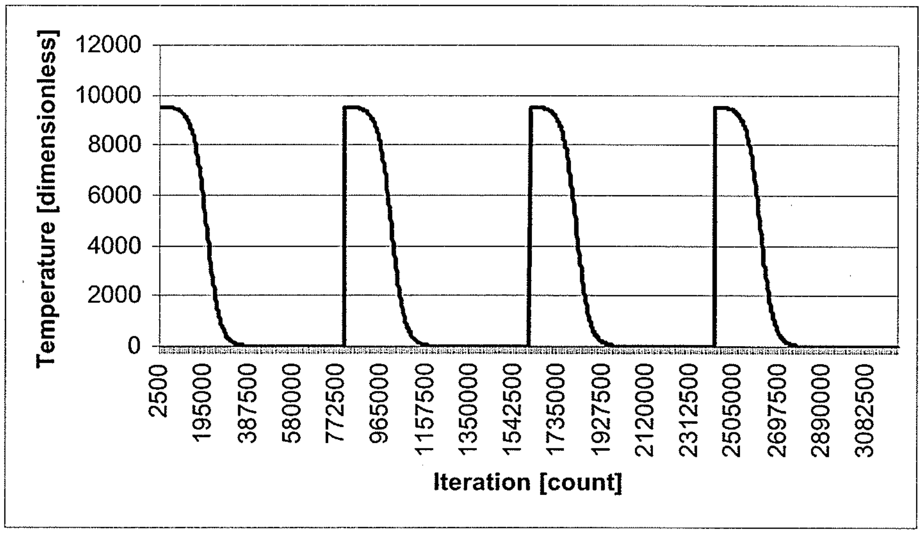


Fig. 5.28 Cooling profile for 2.5 cm tumour.

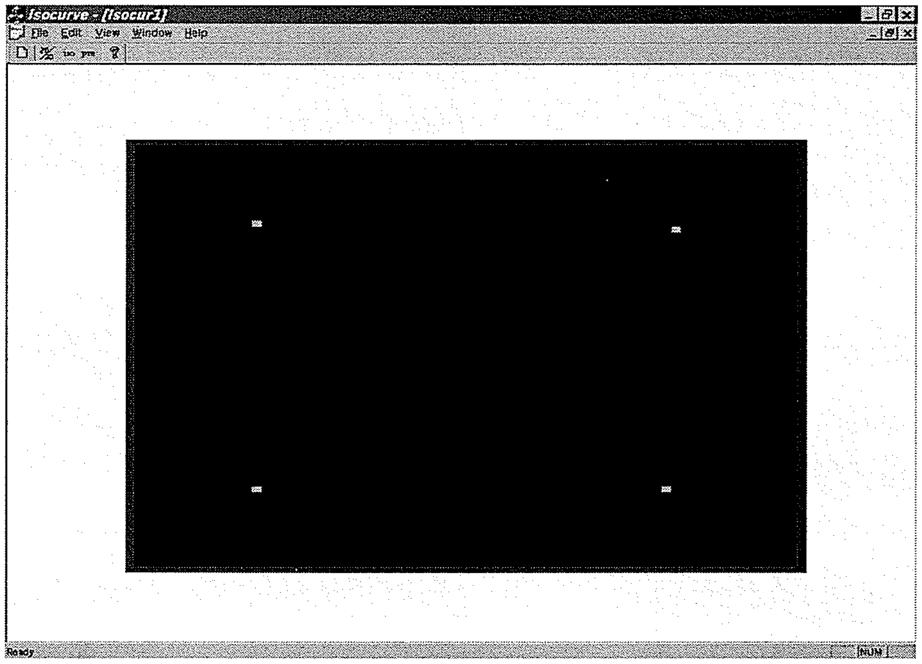


Fig. 5.29 Optimized source positions for 2.5 cm tumour.

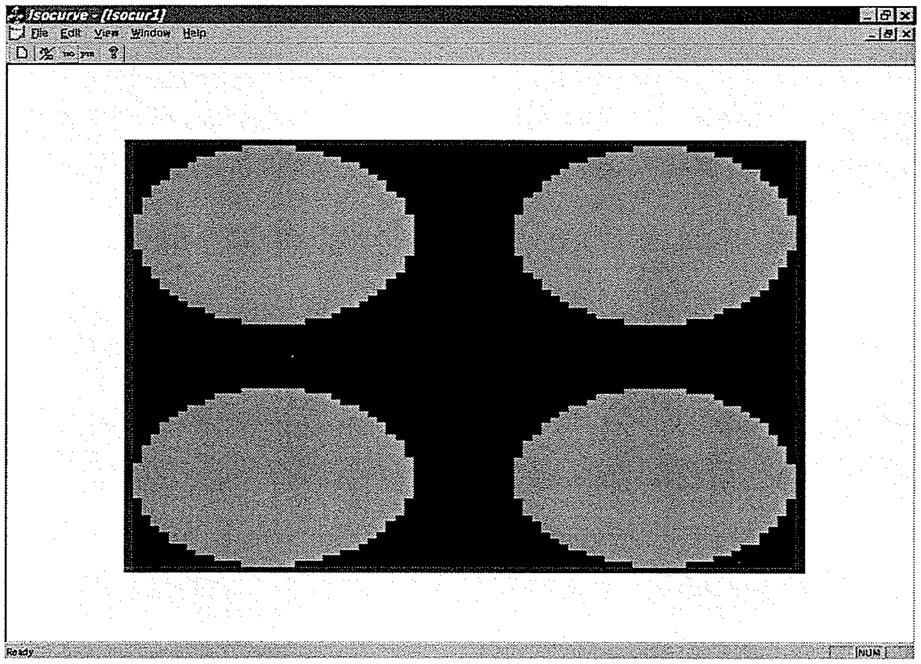


Fig. 5.30 Hyperdose sleeve for 2.5 cm solution.

5.3.6 Square Tumour 3.0 cm in Width

The 3.0 cm tumour shown in Fig. 5.31 was used as input to the SAB software. The configuration file shown in Fig. 5.32 was used to control the flow of the SAB software. This trial produced a cost iteration plot as shown in Fig. 5.33. As specified in previous sections, the cost temperature plot has eight distinct pulses corresponding to the 8 sources being added. A cooling profile shown in Fig. 5.34 was utilized.

In order to illustrate the progression of sources added by the SAB software, and the solutions that it generates, all of the source configurations generated by SAB for the 3.0 cm square tumour are presented.

5.3.6.1 One Source in 3.0 cm Tumour

The first source configuration is the optimal solution for a single source which as shown in Fig. 5.35 has a single source at the center of the tumour. A single source at the center is what we would expect since it would produce the most homogeneous dose in the tumour. However, the hyperdose sleeve for a single source is much to large as shown in Fig. 5.36 where the hyperdose sleeve is as wide as the tumour, 3 cm in diameter.

5.3.6.2 Two Sources in 3.0 cm Tumour

Next SAB will optimize two sources and generates the solution shown in Fig. 5.37. The results are what we would expect to see, a very symmetric distribution of the two sources, maintaining as homogeneous a dose as possible with only two sources, but again a hyperdose sleeve too large, over 3 cm, as shown in Fig. 5.38.

5.3.6.3 Three Sources in 3.0 cm Tumour

Next we add another source into the simulation, and achieve the source placements shown in Fig. 5.39. However, the hyperdose sleeves for three sources is also too large as shown in Fig. 5.40.

5.3.6.4 Four Sources in 3.0 cm Tumour

Next we add another source into the simulation bringing the total number of sources to four. As we can see in Fig. 5.41, a symmetric geometry again is achieved. Unlike the final solutions for the 1.5 cm, 2.0 cm and 2.5 cm square tumours, four sources in a 3.0 cm square tumour is not sufficient as the hyperdose sleeves are too large as shown in Fig. 5.42. Although the hyperdose sleeves shown in Fig. 5.42 seem to be sufficiently small, they are actually 1.1333 cm in diameter (the maximum acceptable being 1.0 cm). This increase in size actually corresponds to an area that is 28.4% larger than the maximum hyperdose sleeve area. Thus the SAB software will continue looking for a better solution by adding another source bringing the total count up to five.

5.3.6.5 Five Sources in 3.0 cm Tumour

The SAB software produces the output shown in Fig. 5.43 for five sources. When the hyperdose sleeve for the distribution shown in Fig. 5.43 is evaluated, it encompasses the majority of the tumour. The previous four SAB solutions continued to make the hyperdose sleeves smaller. Now, we have a situation in which the hyperdose sleeve is getting larger. Although this may appear the be a problem, it is actually a perfect example of why SA is so useful. We are in the process of escaping a local minimum on the search for the global minimum. Five sources can not produce an acceptable dose distribution for a 3.0 cm square tumours as shown by the large hyperdose sleeves in Fig. 5.44, so the SAB software must continue to search by adding another source.

5.3.6.6 Six Sources in 3.0 cm Tumour

Upon inspecting the output for six sources an interesting trend is starting to emerge. It would seem that the solution for five sources is the solution for a single source encompassed within the solution for four sources. This trend continues with the solution for six sources being a rotated solution for two sources encompassed within the solution for four sources as shown in Fig. 5.46. Unfortunately the hyperdose sleeves still encompasses much of the tumour with six sources as shown in Fig. 5.46, and SAB will continue to execute trying seven sources.

5.3.6.7 Seven Sources in 3.0 cm Tumour

The trend continues with seven sources having an optimized source arrangement similar to the optimized output of three sources, encompassed within the optimized solution of four sources. Again this source arrangement produces hyperdose sleeves much too large as shown in Fig. 5.48.

5.3.6.8 Eight Sources in 3.0 cm Tumour

The SAB software continues one last time with eight sources, producing the output shown in Fig. 5.49. The hyperdose sleeve for the solution shown in Fig. 5.49, is shown in Fig. 5.50. As is seen in Fig. 5.50 there are a few points at which the hyperdose sleeves touch, thus making the total area of that hyperdose sleeve too large. However, there are a few explanations for why this is still an optimal solution. The first is that most of the hyperdose sleeves is considerably less than the maximum area, so if it isn't for those few points at which one hyperdose sleeve touches another, this would be a very good solution. The second fact is that the points at which the sleeves join is very small and if it isn't for the quantization of the solution space into pixels, they may not even join. The final cost function value for this solution was 1.052885.

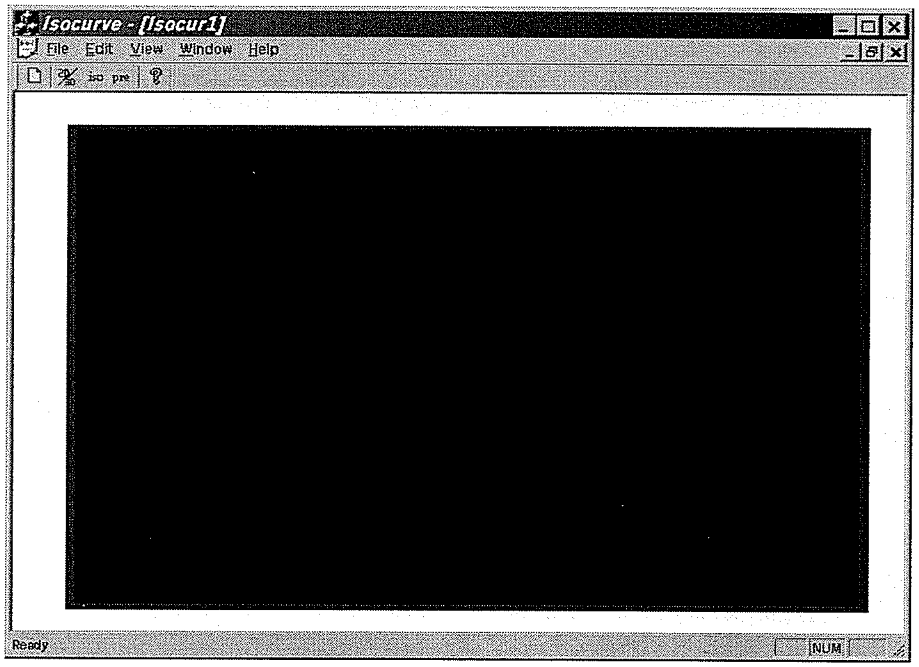
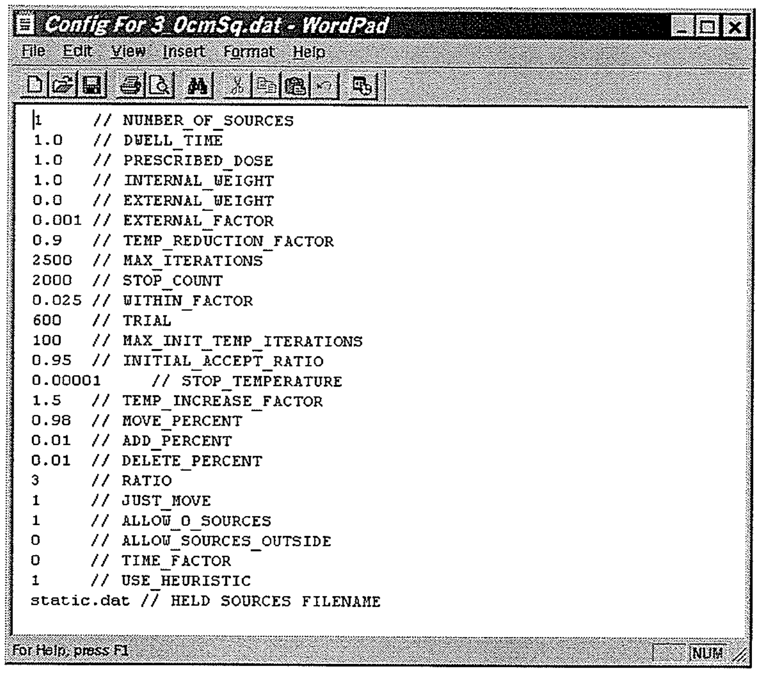


Fig. 5.31 3.0 cm tumour to be optimized with SAB.


Config For 3_0cmSq.dat - WordPad
File Edit View Insert Format Help
For Help, press F1
NUM

```
1      // NUMBER_OF_SOURCES
1.0    // DWELL_TIME
1.0    // PRESCRIBED_DOSE
1.0    // INTERNAL_WEIGHT
0.0    // EXTERNAL_WEIGHT
0.001 // EXTERNAL_FACTOR
0.9    // TEMP_REDUCTION_FACTOR
2500   // MAX_ITERATIONS
2000   // STOP_COUNT
0.025 // WITHIN_FACTOR
600    // TRIAL
100    // MAX_INIT_TEMP_ITERATIONS
0.95   // INITIAL_ACCEPT_RATIO
0.00001     // STOP_TEMPERATURE
1.5    // TEMP_INCREASE_FACTOR
0.98   // MOVE_PERCENT
0.01   // ADD_PERCENT
0.01   // DELETE_PERCENT
3      // RATIO
1      // JUST_MOVE
1      // ALLOW_0_SOURCES
0      // ALLOW_SOURCES_OUTSIDE
0      // TIME_FACTOR
1      // USE_HEURISTIC
static.dat // HELD SOURCES FILENAME
```

Fig. 5.32 Configuration file for 3.0 cm tumour.

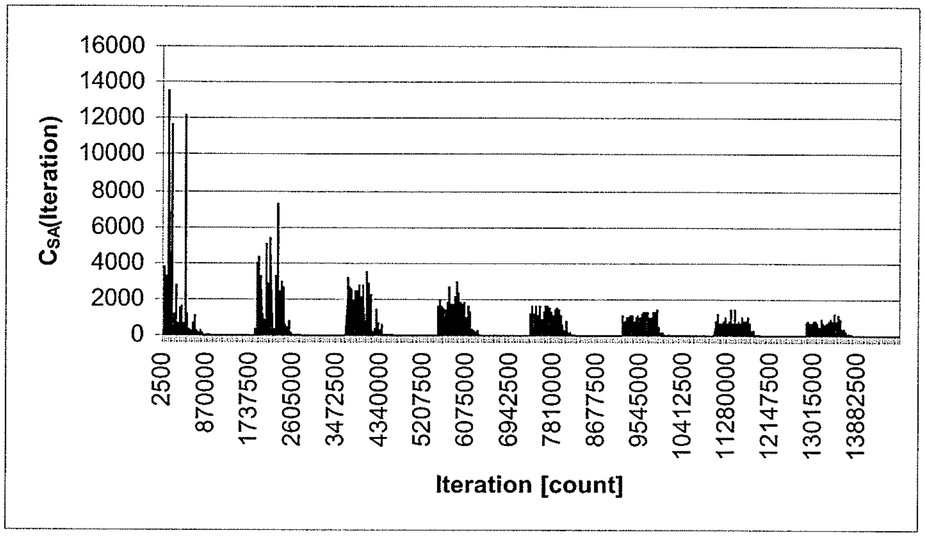


Fig. 5.33 Cost plot for 3.0 cm tumour.

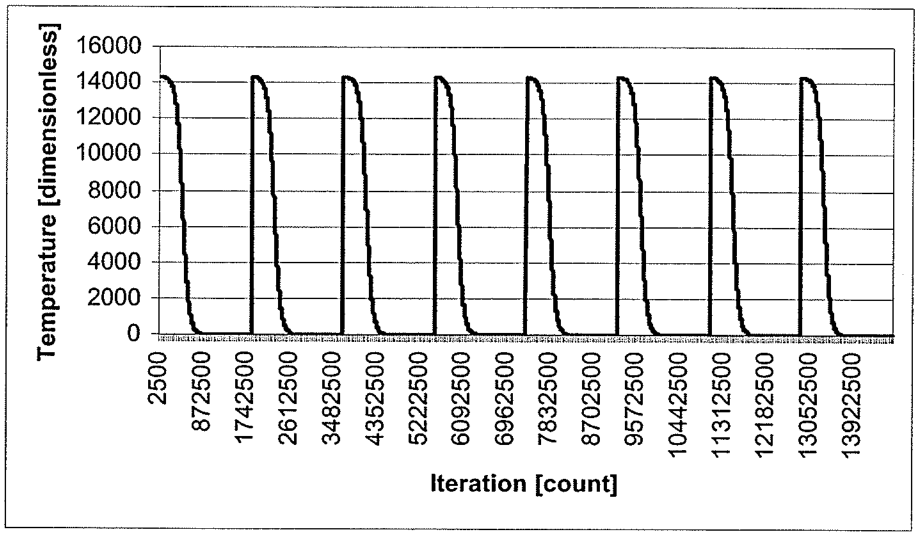


Fig. 5.34 Cooling profile for 3.0 cm tumour.

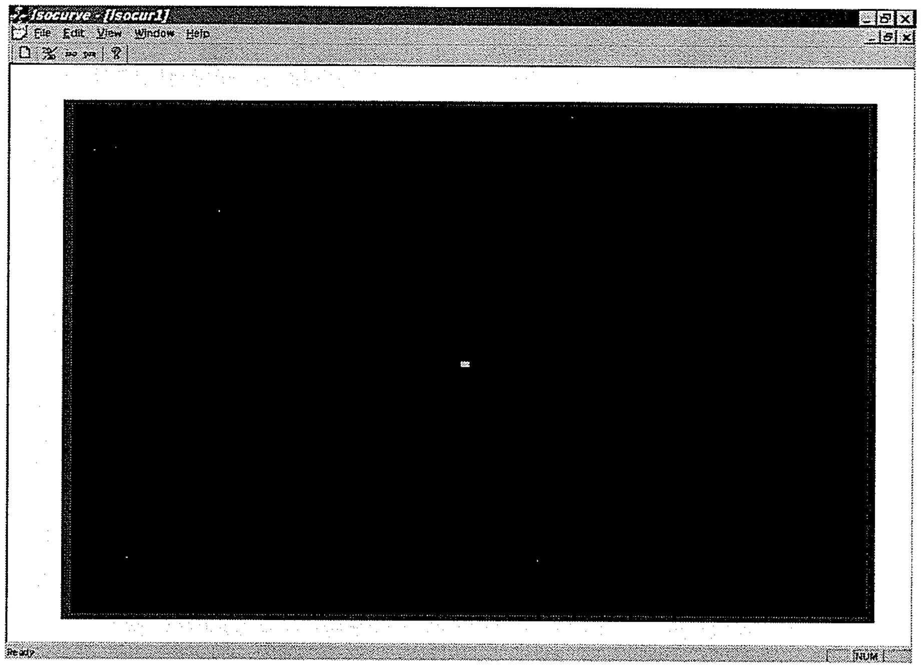


Fig. 5.35 Single source position in a 3.0 cm square tumour.

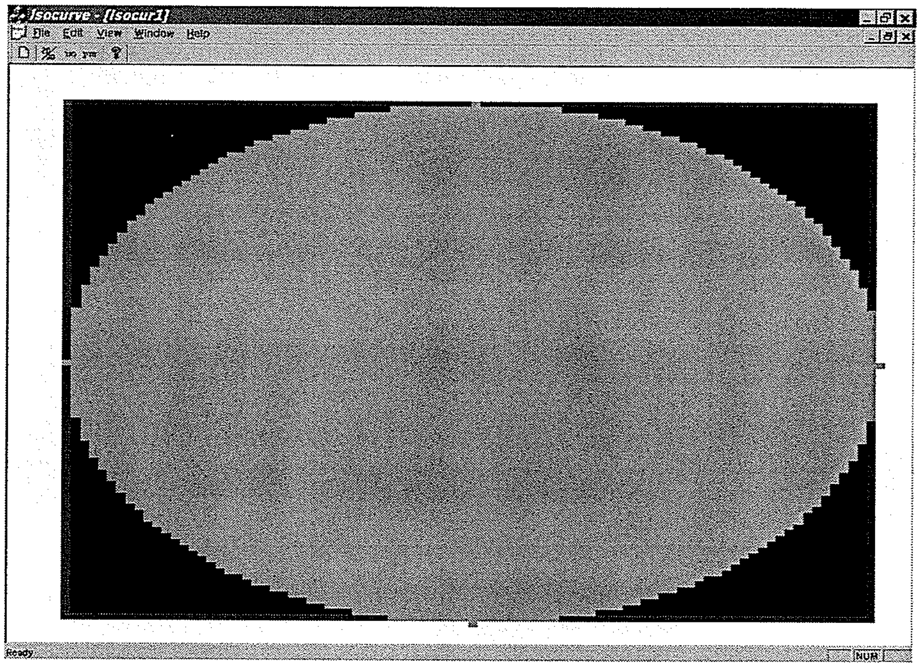


Fig. 5.36 Hyperdose sleeve for single source in 3.0 cm tumour.

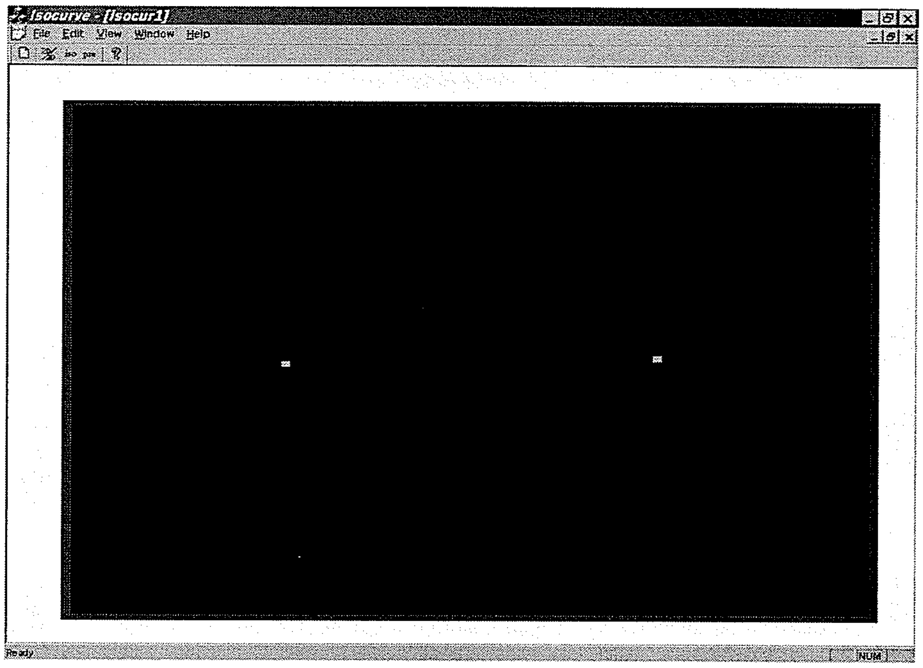


Fig. 5.37 Optimized solution of two sources in a 3.0 cm square tumour.

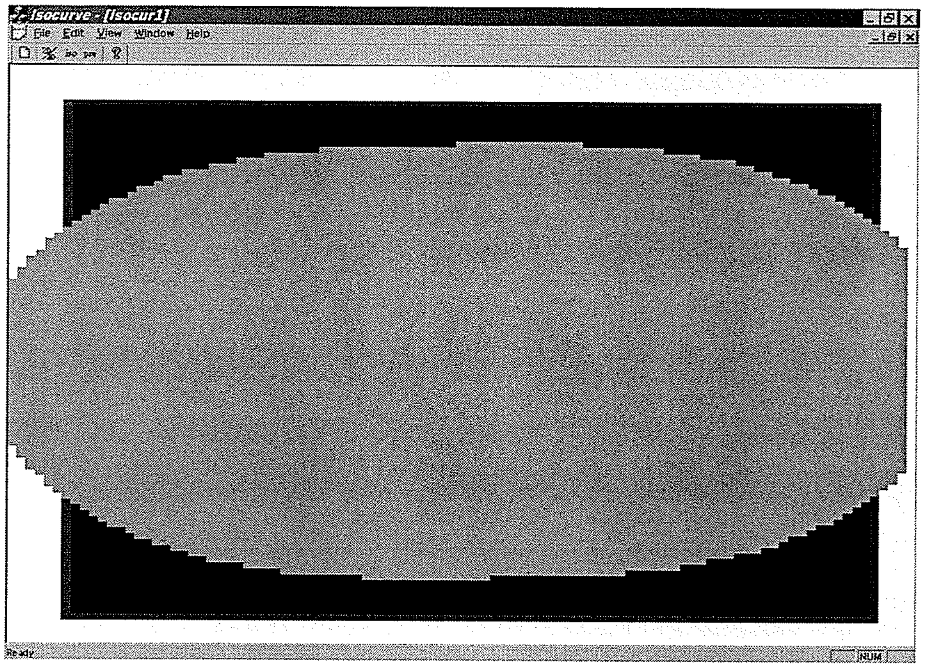


Fig. 5.38 Hyperdose sleeve for two sources in a 3.0 cm square tumour.

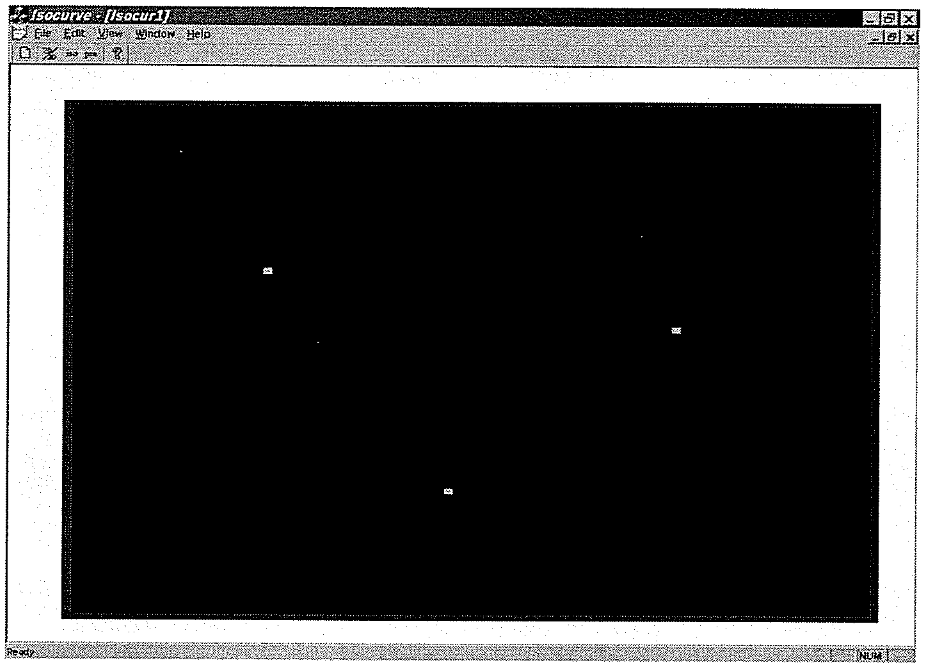


Fig. 5.39 Optimized solution of three sources in a 3.0 cm square tumour.

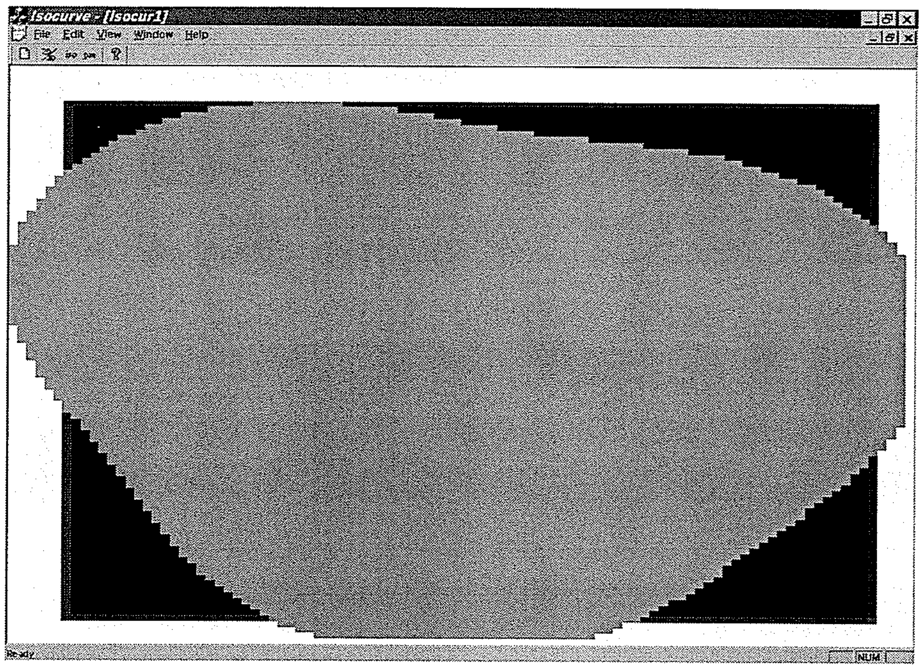


Fig. 5.40 Hyperdose sleeve of three sources in a 3.0 cm square tumour.

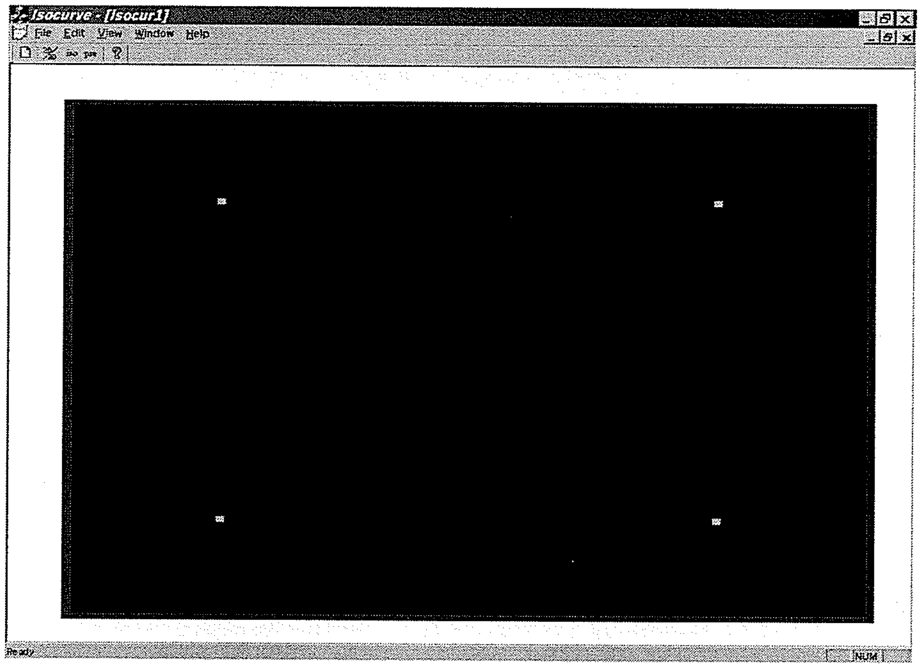


Fig. 5.41 Optimized solution of four sources in a 3.0 cm square tumour.

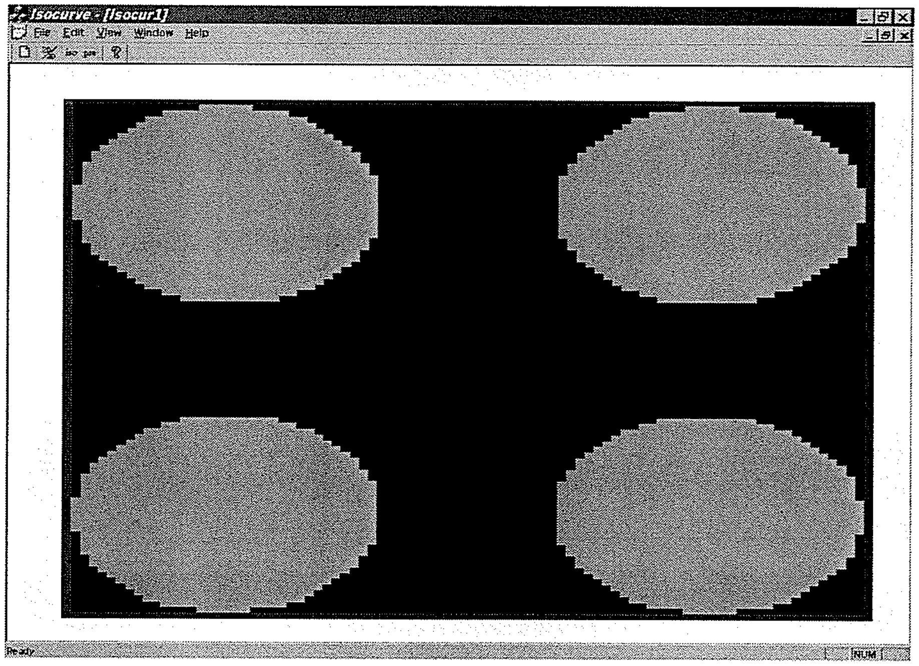


Fig. 5.42 Hyperdose sleeve for four sources in a 3.0 cm square tumour.

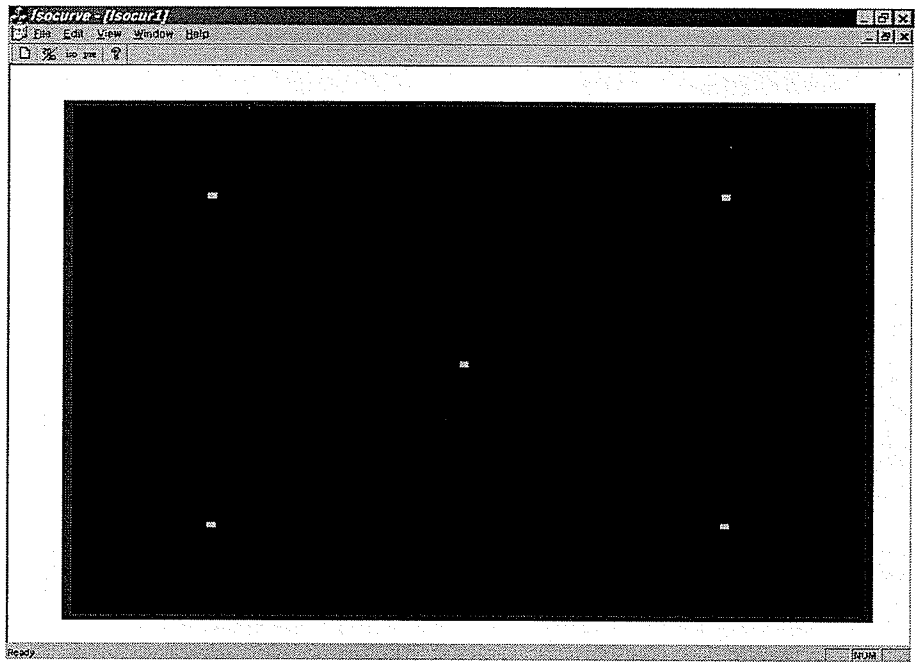


Fig. 5.43 Optimized solution of five sources in a 3.0 cm square tumour.

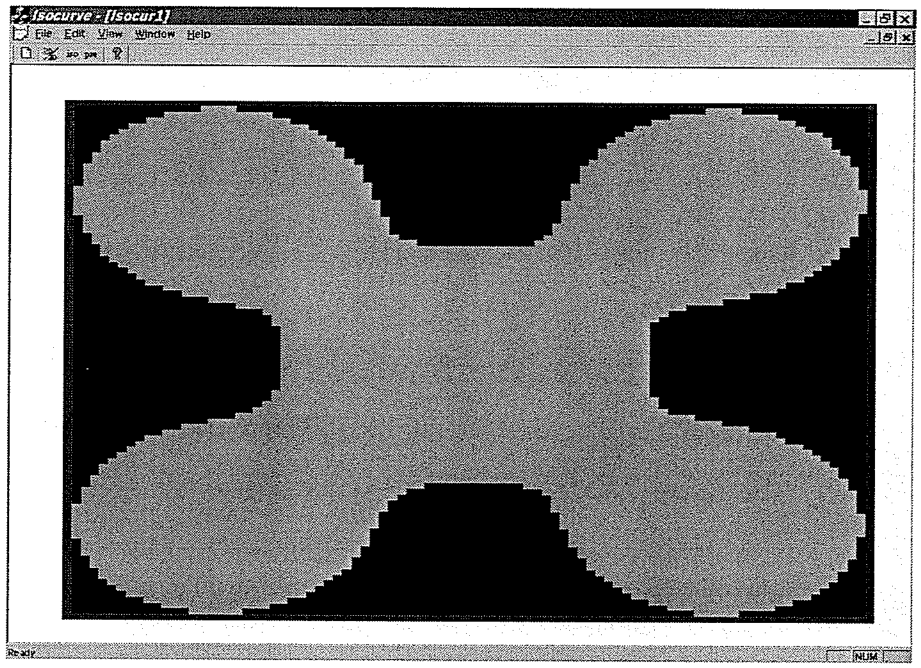


Fig. 5.44 Hyperdose sleeve of five sources in a 3.0 cm square tumour.

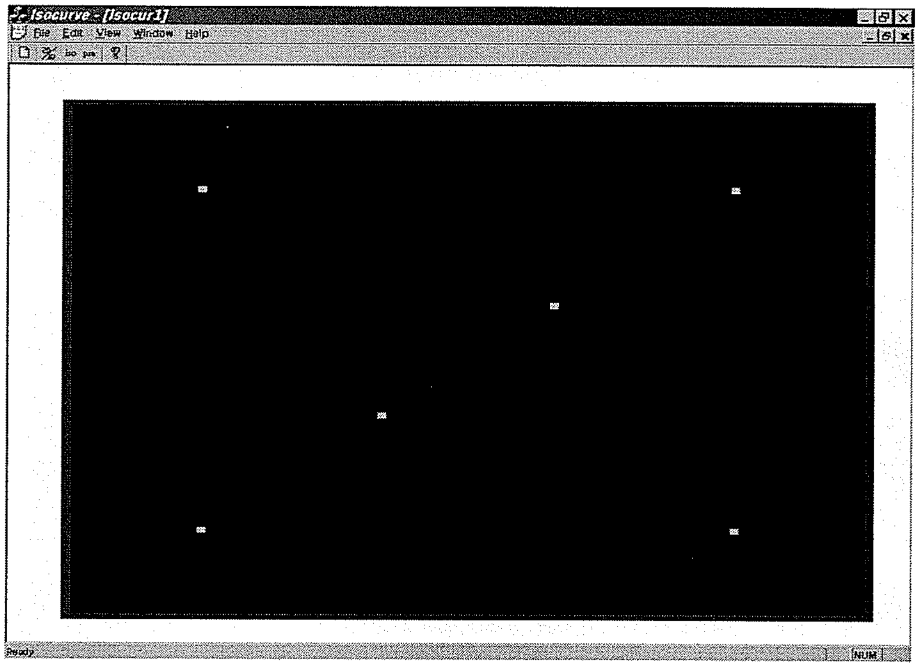


Fig. 5.45 Optimized solution of six sources in a 3.0 cm square tumour.

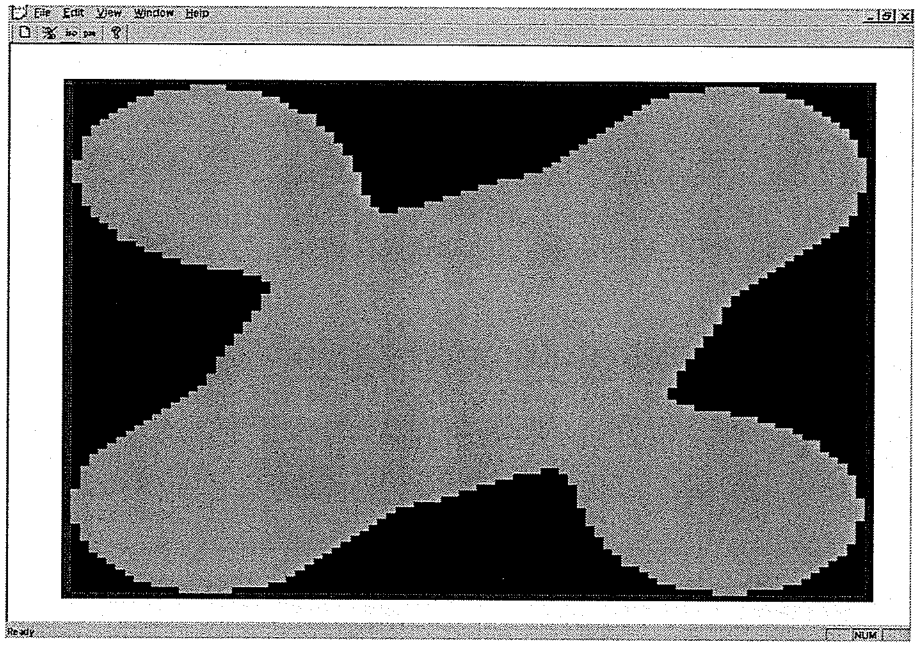


Fig. 5.46 Hyperdose sleeve of six sources in a 3.0 cm square tumour.

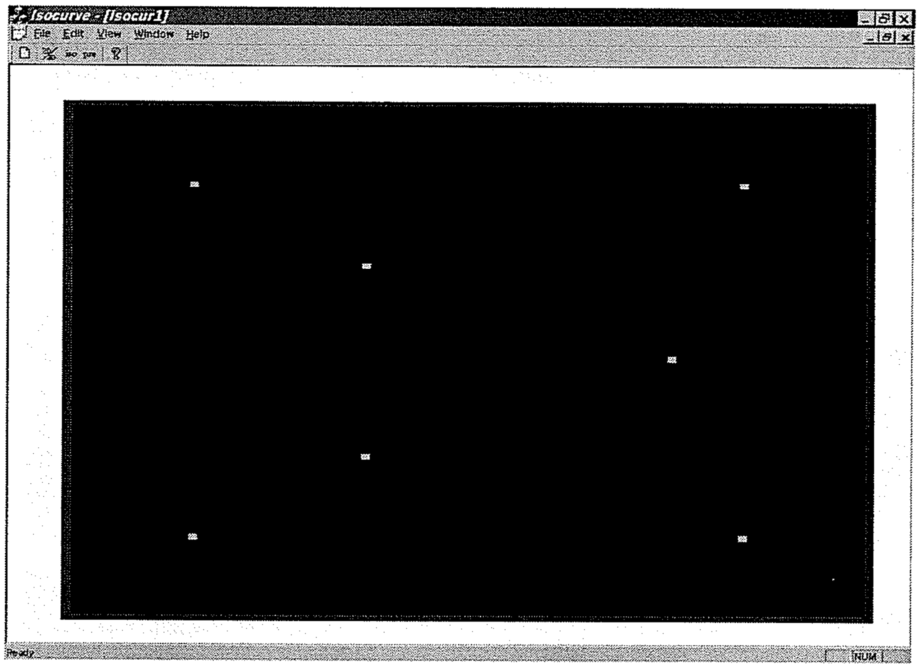


Fig. 5.47 Optimized solution of seven sources in a 3.0 cm square tumour.

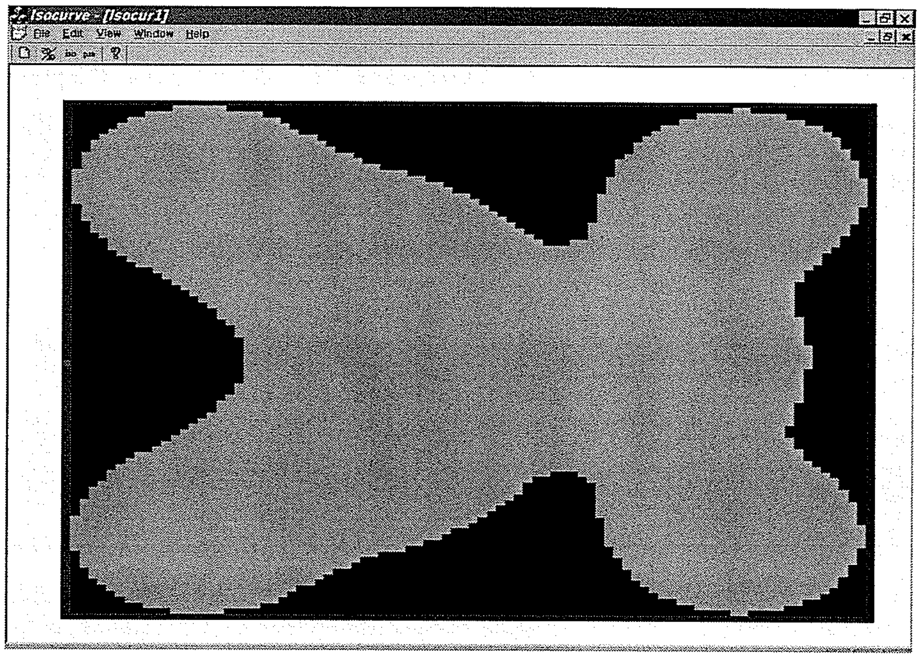


Fig. 5.48 Hyperdose sleeve of seven sources in a 3.0 cm square tumour.

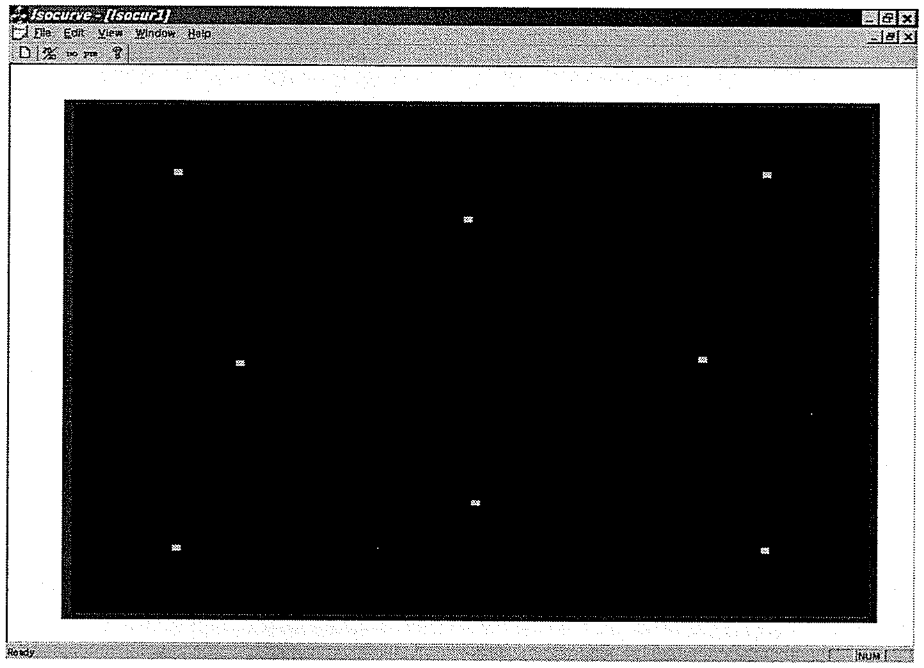


Fig. 5.49 Optimized source positions for 3.0 cm tumour.

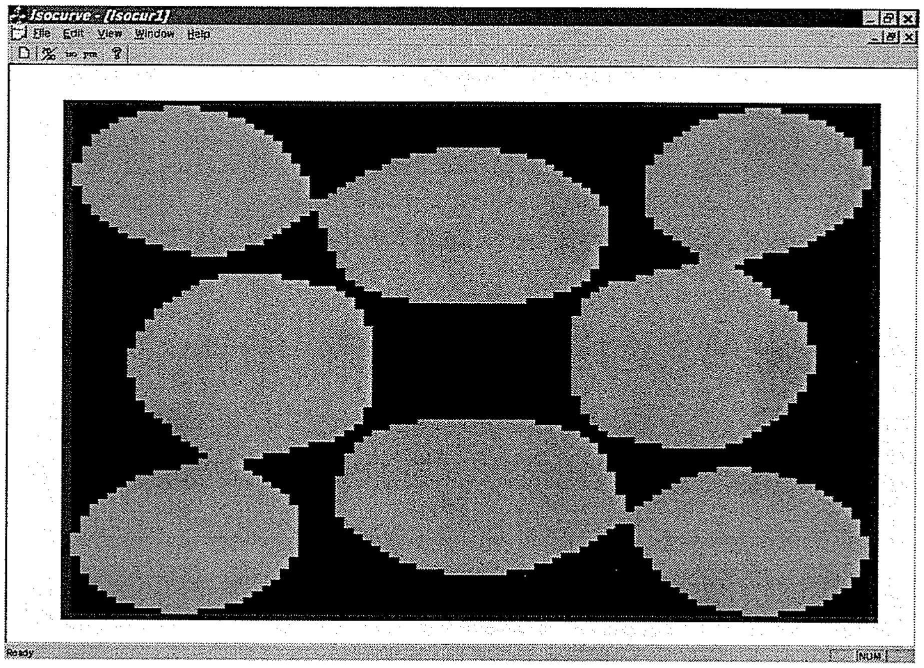


Fig. 5.50 Hyperdose sleeve for 3.0 cm solution.

5.4 Conclusions for SA

The results for the different tumour shapes presented in the proceeding section indicate that we have implemented a SA algorithm and software program (SAB) that can optimize source positions in square tumours. Additional results for other tumour sizes are presented in Appendix A. The following section will now take the results from SAB and use them to train an ANN to optimize brachytherapy source placement in 2D square tumours.

5.5 Experimental Design for ANN

The goal of this thesis is to develop an ANN for brachytherapy optimization. In order to develop the ANN it is required to determine what inputs and outputs to use to optimize brachytherapy implants as well as, the best structure of the ANN, and in doing so, prove that ANNs can be used for brachytherapy optimization. Section 5.6 presents the experimental results with different forms of input to an ANN and determines which form is the best. The data gathered with SAB as presented in previous sections is used as input data. The training data producing the ANN that performs best on the test data will be considered the best form of input.

5.6 Experimental Results and Discussion for ANN

The first set of experiments will be used to determine what SAB output to use to train an ANN to optimize brachytherapy implants (covered in Section 5.6.1). Once the training data has been determined, the best form of ANN (number of input, internal and output nodes) will be determined as presented in Section 5.6.2.

5.6.1 Training data for ANN optimization

In order to simplify the design of the training data, all of the outputs from SAB are normalized so that they are the same size in terms of pixels. For example the 1.0 cm tumour would be expanded to be the same pixel size as the 3.0 cm tumour. The 1.0 cm tumour then has three times the number of pixels per mm than the 3.0 cm tumour.

5.6.1.1 Training with source positions

Once the tumours were normalized to a common size, the first form of input investigated was to simply use the positions of the point sources and a measure of the size of the tumour. For instance the 1.0 cm square tumour in Section 5.3.2 would have an input of 18,18,0.33333 corresponding to x and y coordinates for the single point source of 18 and 18 respectively, and a size of 3 pixels per mm. The 3.0 cm tumour from Section 5.3.6 would have the following input 7,7,8,9,8,30,19,8,19,28,30,8,28,19,30,30,1.0 corresponding to the x and y coordinates of all 8 sources respectively, followed by the size parameter (ratio) of 1.0 signifying a 1 pixel per mm ratio. As some tumours have more point sources than others, the ANN will be designed to have enough input and output nodes to accommodate the tumour with the most point sources - a 3.0 cm square. Tumours with less point sources will have nodes that do not get input. A value of -1 will be used to signify that a node has no input. The 1.0 cm, 2.0 cm and 3.0 cm square tumours are used as input, and the other tumours are used as test data. A summary of the input used is shown in Table 5.1, Table 5.2,and Table 5.3.

Table 5.1: ANN input using coordinates for 1.0 cm square tumour.

Node	Input	Desired Output
1 (x_1)	-1 or 18	18
2 (y_1)	-1 or 18	18
3 (x_2)	-1	-1
4 (y_2)	-1	-1

Table 5.1: ANN input using coordinates for 1.0 cm square tumour.

Node	Input	Desired Output
5 (x_3)	-1	-1
6 (y_3)	-1	-1
7 (x_4)	-1	-1
8 (y_4)	-1	-1
9 (x_5)	-1	-1
10 (y_5)	-1	-1
11 (x_6)	-1	-1
12 (y_6)	-1	-1
13 (x_7)	-1	-1
14 (y_7)	-1	-1
15 (x_8)	-1	-1
16 (y_8)	-1	-1
17	0.333333	NONE

Table 5.2: ANN input using coordinates for 2.0 cm square tumour.

Node	Input	Desired Output
1 (x_1)	-1 or 9	9
2 (y_1)	-1 or 9	9
3 (x_2)	-1 or 9	9
4 (y_2)	-1 or 28	28

Table 5.2: ANN input using coordinates for 2.0 cm square tumour.

Node	Input	Desired Output
5 (x_3)	-1 or 28	28
6 (y_3)	-1 or 9	9
7 (x_4)	-1 or 28	28
8 (y_4)	-1 or 28	28
9 (x_5)	-1	-1
10 (y_5)	-1	-1
11 (x_6)	-1	-1
12 (y_6)	-1	-1
13 (x_7)	-1	-1
14 (y_7)	-1	-1
15 (x_8)	-1	-1
16 (y_8)	-1	-1
17	0.666666	NONE

Table 5.3: ANN input using coordinates for 3.0 cm square tumour.

Node	Input	Desired Output
1 (x_1)	-1 or 7	7
2 (y_1)	-1 or 7	7
3 (x_2)	-1 or 8	8
4 (y_2)	-1 or 19	19

Table 5.3: ANN input using coordinates for 3.0 cm square tumour.

Node	Input	Desired Output
5 (x_3)	-1 or 7	7
6 (y_3)	-1 or 30	30
7 (x_4)	-1 or 19	19
8 (y_4)	-1 or 8	8
9 (x_5)	-1 or 19	19
10 (y_5)	-1 or 28	28
11 (x_6)	-1 or 30	30
12 (y_6)	-1 or 8	8
13 (x_7)	-1 or 28	28
14 (y_7)	-1 or 19	19
15 (x_8)	-1 or 30	30
16 (y_8)	-1 or 30	30
17	0.666666	NONE

When a BP ANN with 17 input nodes, 4 hidden layer nodes, and 16 output nodes is trained with the training data in Table 5.1, Table 5.2, and Table 5.3 it achieves an RMS error on the training data of only 0.445589. However, on a test set of other tumour shapes it produces an RMS error of 2.797090 from the desired outputs. When the output is inspected visually, a number of problems are identified. For example too many sources being used for smaller tumour sizes. Another issue that became apparent with more experimentation is that the output was very dependent on the training data. In other words, the network was not able to generalize. For instance, if a source in the input data was shifted by a single pixel value (i.e. from an x value of 10 mm to an x value of 11 mm) the output would have a source at both pixel locations or many extra sources in the tumour. From the results obtained with this network configuration, it was decided that a different form of input with more information was required.

5.6.1.2 Training with tumour images from SAB

The next form of input that was used to create training data for an ANN was the tumour images obtained from SAB. For instance the tumour shown in Fig. 5.49, was used as input to an ANN. The inputs were the pixel values in the tumour (as described in Table 4.1) without the sources. The desired outputs were the pixel values in the tumour with the sources. This form of training vector was an attempt at training an ANN to literally learn the positions of the sources. Similar results as those obtained with the training vectors in the previous section, were achieved. The major problem again with this form of training data was the inability of the network to generalize if a source was shifted slightly from a position that the network was trained to recognize.

5.6.1.3 Training with energy distribution

After the initial attempts at designing an ANN to optimize brachytherapy cancer treatments, it was realized that an input that is continuous is better than an input that is discrete. If the sources in the tumour are used to calculate the dose at each point in the tumour we can achieve a nearly continuous form of input. The dose at each point in the tumour is considered to be the energy distribution in the tumour and due to this nearly continuous nature would make a good form of training data for an ANN. For example, the input could be the current energy distribution in the tumour (based on the sources present), and the output could be the desired energy distribution (based on the optimal source locations found with SAB). The only discontinuity in the energy distribution are at the exact point source locations (where the dose is essentially infinite). However, if the energy distribution is modified such that they are continuous at these points as well, then there is a continuous form of input and output to use. Another advantage to this form of input and output is that small shifts in the point sources should not affect the output as much. In the previous forms of input, their discrete nature had an adverse effect because a source was either present or absent (a binary 0 or 1 scenario), making it very hard for the ANN to generalize. If the energy distribution is used as input, a source shifted slightly will still have significantly elevated energy in the region, and the scenario is no longer binary.

In order to use the energy distribution as input and output, a large number of input and output nodes are required. As the tumours are normalized to be thirty pixels by thirty pixels, it requires nine hundred input and output nodes (30 x 30). An additional input will also be required to specify the ratio of the input. Thus a total of 901 input nodes and 900 output nodes are required.

In preparing the training data, a few modifications have to be made to the output from the SAB program. In addition to normalization of the tumour size (discussed at the beginning of this Section) the energy distribution is also normalized. As covered in Section 2.5.9, weights in a BP ANN are adjusted during training, according to Eq. 2.7. If the difference between the desired weight (T) and achieved output (Y) is large, then the weight change value will become large. In theory this is not a problem. However, when actually implementing these formulas in software, the values of the variables must be stored in RAM. Thus, if the numbers become too large, round off errors and overflow errors occur on the variables. In order to reduce the possibility of this occurring, the training data will be modified to normalize the energy distribution to values between 0.0 and 1.0 by dividing all of the values by the maximum. A sample form of training data input for the SAB solution shown in Fig. 5.49, is shown in Fig. 5.51. The output from an ANN trained with the energy distribution will be the optimal energy distribution in the input tumour. As it is the source placement within the tumour that is of importance, a threshold function is used to find the peaks in the output energy distribution. The peaks will be representative of where a source should be placed. It is important to note that the error calculated during the training and testing is the RMS error on the energy function, not on the actual source placement. Therefore, a large RMS error may not have a drastic effect on the actual source location. This is because if every point in the energy distribution is incorrect by a small amount the peaks will still be nearly correctly placed thus the derived source positions will still be correct, but the difference between the desired energy distribution and the predicted energy (the RMS error) will be potentially large.

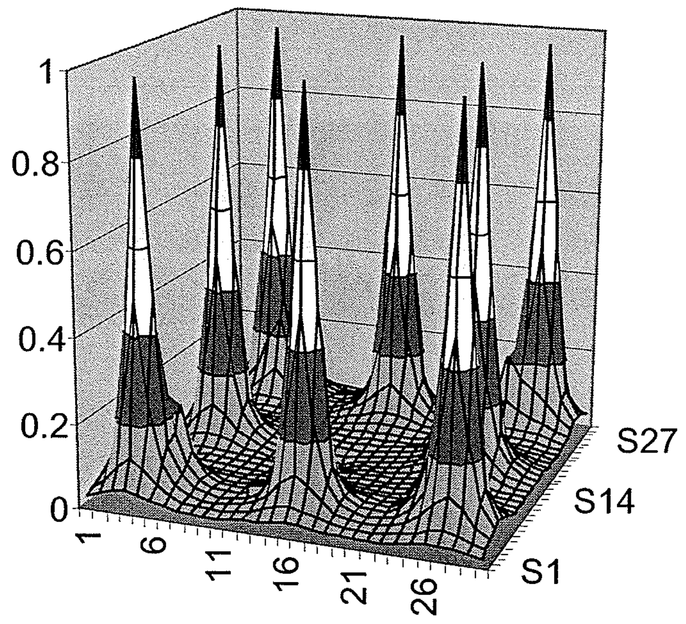


Fig. 5.51 Sample of energy distribution input to ANN for training.

Additive noise was also used on the training data as discussed in Section 2.5.9.1. The source configurations described in Table 5.4 were used to create the training data.

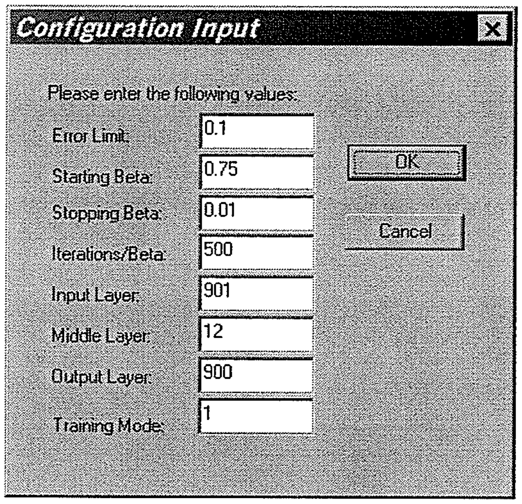
Table 5.4: Input used for energy distribution training data.

Tumour Size	Ratio	Source Locations (pixels - x, y)	Additive Noise Max Distance (pixels)	Number of Samples
1.0 cm	0.333333	NONE	3	10
1.0 cm	0.333333	18,18	3	10
2.0 cm	0.666666	NONE	4	10
2.0 cm	0.666666	9,9	4	10
2.0 cm	0.666666	9,28	4	10
2.0 cm	0.666666	28,9	4	10
2.0 cm	0.666666	28,28	4	10
2.0 cm	0.666666	9,9 9,28 28,9 28,28	4	10
3.0 cm	1.0	NONE	5	10
3.0 cm	1.0	7,7	5	10
3.0 cm	1.0	8,19	5	10
3.0 cm	1.0	8,30	5	10
3.0 cm	1.0	19,8	5	10
3.0 cm	1.0	19,28	5	10

Table 5.4: Input used for energy distribution training data.

Tumour Size	**Ratio**	**Source Locations (pixels - x, y)**	**Additive Noise Max Distance (pixels)**	**Number of Samples**
3.0 cm	1.0	30,8	5	10
3.0 cm	1.0	28,19	5	10
3.0 cm	1.0	30,30	5	10
3.0 cm	1.0	7,7 8,19 8,30 19,8 19,28 30,8 28,19 30,30	5	10

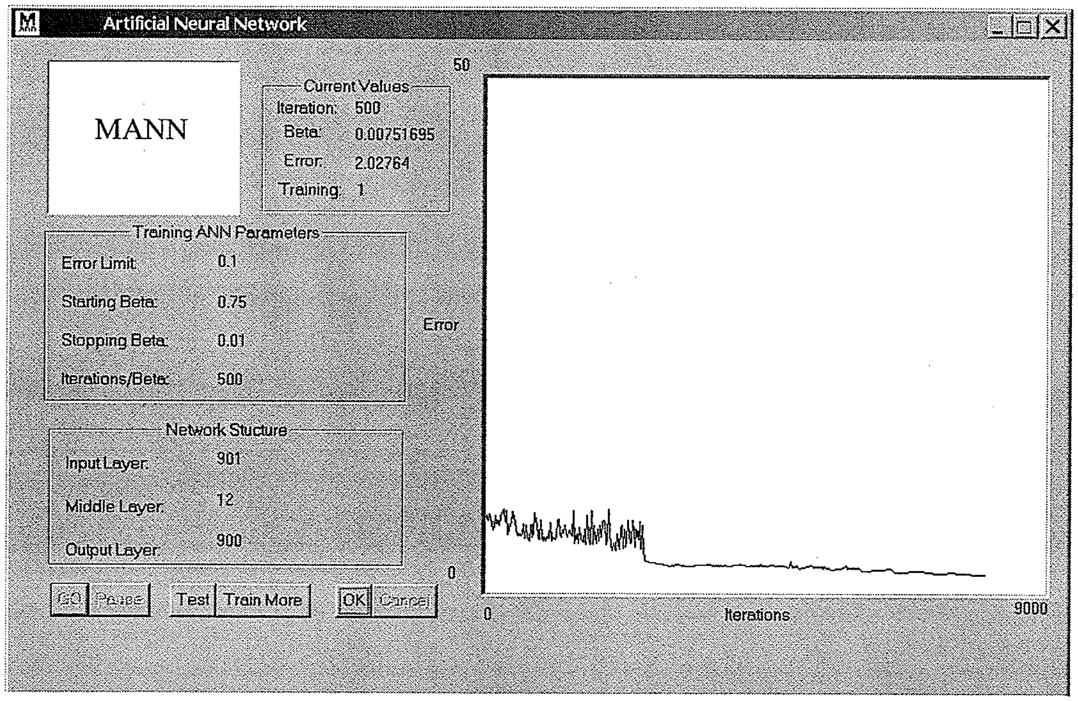
A sample input to the MANN software is shown in Fig. 5.52, with the corresponding output from the MANN software shown in Fig. 5.53.



Please enter the following values:

Error Limit:	0.1
Starting Beta:	0.75
Stopping Beta:	0.01
Iterations/Beta:	500
Input Layer:	901
Middle Layer:	12
Output Layer:	900
Training Mode:	1

Fig. 5.52 Input for MANN software with 12 internal nodes.



Current Values

Iteration:	500
Beta:	0.00751695
Error:	2.02764
Training:	1

Training ANN Parameters

Error Limit:	0.1
Starting Beta:	0.75
Stopping Beta:	0.01
Iterations/Beta:	500

Network Stucture

Input Layer:	901
Middle Layer:	12
Output Layer:	900

Fig. 5.53 Results of training ANN using MANN for 12 internal nodes.

Additional results from the MANN software for different numbers of internal nodes can be found in Appendix B. The following section will cover the validation of the ANN developed with the energy function as input, as well as determine the optimal number of internal nodes.

5.6.2 Finding the Best Form of BP ANN for Brachytherapy Optimization

In order to validate the ANN trained with the energy distribution the techniques discussed in Section 2.5.10 will be used. All of the data obtained with SAB will be separated into two sets, a training set and a test set. The training set will be used to train the ANN and the test set will be used to see how well the ANN can perform on data that it has not been trained on. The training and test set will be divided as shown in Table 5.5.

Table 5.5: Division of SAB data into training and test sets.

Tumour Size	**Allocated Set**
0.5 cm	Test
0.8 cm	Test
1.0 cm	Training
1.2 cm	Test
1.5 cm	Test
1.8 cm	Test
2.0 cm	Training

Table 5.5: Division of SAB data into training and test sets.

Tumour Size	**Allocated Set**
2.2 cm	Test
2.5 cm	Test
2.8 cm	Test
3.0 cm	Training

In order to determine the optimal number of interior nodes to use for brachytherapy optimization, a cost function will be used. The cost function is

$$c_{ann} = a\left(\frac{(n_i - n_{min})}{n_{max}}\right) + b\left(\frac{e_n}{e_{max}}\right) \tag{5.1}$$

where a and b are weights that assign the level of importance of the two terms, n_i is the number of internal nodes, n_{max} is the maximum number of internal nodes we will allow and is defined as 20, n_{min} is defined as 2 because a BP ANN must have at least two internal nodes, e_n is the error on the test set, and e_{max} is the maximum error which we get with only 2 internal nodes and is defined as 30. It was decided that a will be 0.25 and b will be 0.75, indicating that we are more concerned with the error achieved on the test set, rather than the number of internal nodes. The ANN that has the smallest value for c_{amn} (Eq. 5.1) will be considered the best ANN for brachytherapy optimization as it is a measure of generalization versus memorization. Using the results from SAB and Eq. 5.1, the results for the various ANN's is shown in Table 5.6.

Table 5.6: Results of tests for different number of internal nodes.

Number of Internal Nodes	RMS Error on Test Set	c_{ann} (Eq. 5.1)	Predicted Maximum Positional Error (mm)
6	27.74	0.74	8.32
8	20.29	0.58	6.09
10	15.75	0.49	4.73
12	13.37	0.46	4.01
14	12.99	0.48	3.90
16	12.96	0.50	3.89
18	12.87	0.52	3.86

Based on the results in Table 5.6, an ANN with 12 internal nodes will be best for optimizing brachytherapy treatments when using the energy distribution in the tumour as the input and output from the ANN.

5.6.3 Evaluation of ANN with 12 Internal Nodes

Now that the number of internal nodes has been chosen, the output from that ANN will be evaluated to see how well it does on the training data. As shown in Table 5.6, the ANN with 12 internal nodes has an RMS error of 13.37 on the test set, but it is required to evaluate were this error is incurred. Therefore, all of the test sets identified in Table 5.5

will be evaluated to see how well the ANN performs. In the following sections, all of the tumour positions will be indicated in their transformed positions with their ratio specified.

5.6.3.1 0.5 cm Tumour Predicted Position

A 0.5 cm tumour has a ratio of 0.166666 and a SAB optimized source position of 20,20 for x and y respectively. The output from the ANN produced 19,20 for x and y respectively corresponding to a Euclidian distance of 1 pixel. 1 pixel at this ratio corresponds to 0.16 mm of positional error. The output from the ANN is as expected.

5.6.3.2 0.8 cm Tumour Predicted Position

A 0.8 cm tumour has a ratio of 0.266666 and a SAB optimized source position of 20,20 for x and y respectively. The output from the ANN produced 20,20 for x and y respectively, thus there is no error for this tumour size.

5.6.3.3 1.2 cm Tumour Predicted Position

A 1.2 cm tumour has a ratio of 0.4 and a SAB optimized source position of 20,20 for x and y respectively. The output from the ANN produced 19,20 for x and y respectively corresponding to a Euclidian distance of 1 pixel. 1 pixel at this ratio corresponds to 0.4 mm of positional error. The output from the ANN is as expected.

5.6.3.4 1.5 cm Tumour Predicted Position

A 1.5 cm tumour has a ratio of 0.5 and a SAB optimized source position as shown in Table 5.7.

Table 5.7: Predicted source positions for 1.5 cm tumour.

Desired Positions (pixels) (x, y)	Actual Positions (pixels) (x, y)	Euclidian Distance (pixels)	Error (mm)
12,12	10,11	2.24	1.12
12,29	10,30	2.24	1.12
29,12	29,11	1.00	0.50
29,29	30,30	1.41	0.71

Thus the maximum error for a 1.5 cm tumour is 1.12 mm. This is still very little error and considered acceptable.

5.6.3.5 1.8 cm Tumour Predicted Position

A 1.8 cm tumour has a ratio of 0.6 and a SAB optimized source position as shown in Table 5.8.

Table 5.8: Predicted source positions for 1.8 cm tumour.

Desired Positions (pixels) (x, y)	Actual Positions (pixels) (x, y)	Euclidian Distance (pixels)	Error (mm)
12,12	11,11	1.41	0.85
12,29	10,30	2.24	1.34
29,12	29,10	2.00	1.20
29,29	30,29	1.41	0.60

Thus the maximum error for a 1.8 cm tumour is 1.34 mm. This is still very little error and considered acceptable.

5.6.3.6 2.2 cm Tumour Predicted Position

A 2.2 cm tumour has a ratio of 0.733333 and a SAB optimized source position as shown in Table 5.9.

Table 5.9: Predicted source positions for 2.2 cm tumour.

Desired Positions (pixels) (x, y)	Actual Positions (pixels) (x, y)	Euclidian Distance (pixels)	Error (mm)
12,12	10,10	2.83	2.08
12,29	10,29	2.00	1.47
29,12	29,10	2.00	1.47
29,29	30,30	1.41	1.03

Thus the maximum error for a 2.2 cm tumour is 2.08 mm.

5.6.3.7 2.5 cm Tumour Predicted Position

A 2.5 cm tumour has a ratio of 0.833333 and a SAB optimized source positions as shown in Table 5.10 and Fig. 5.29. However, when the output from the ANN is inspected, it has 8 sources present in the solution as shown in Fig. 5.54. Initially it seemed that there was a flaw in the ANN and that it was producing false positives. However, after further investigating, it was realized that as indicated in Section 5.3.5, the hyperdose sleeve for the SAB optimized source positions are exactly 1.0 cm. Therefore any movement in the source positions would cause the hyperdose sleeves to be too large and thus the source positions would no longer be the optimized positions. Recall that at the beginning of the discussion for the development of the ANN it was indicated that the input to the ANN would be the output from SAB scaled down to be a 30 mm by 30 mm square. Unfortunately at this reduced resolution the source positions are shifted enough that the hyperdose sleeves that were once exactly 1.0 cm in diameter become 1.1 cm in diameter. Thus four sources is not a good solution at this reduced resolution. This is perhaps one of the best indications that the ANN is in fact truly optimizing source positions, as it knew that four sources was insufficient and thus predicted that eight sources were necessary. When the hyperdose sleeve size for the eight source solution are checked, they are only 0.7 cm in diameter which is better than the 1.1 cm diameter that four sources cause. Therefore, although the results do not match what is expected, they are the correct results for this resolution of a 2.5 cm square tumour.

Table 5.10: SAB optimized source positions for 2.5 cm tumour.

Desired Positions
12,12
12,30
30,21
30,30

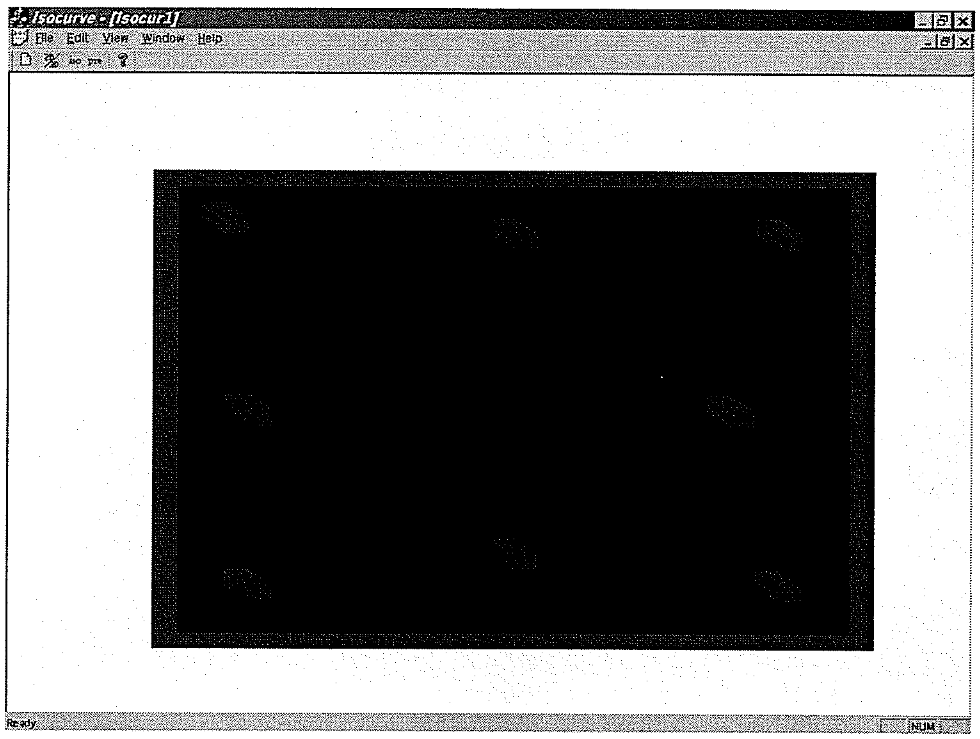


Fig. 5.54 Predicted source positions for 2.5 cm tumour.

5.6.3.8 2.8 cm Tumour Predicted Position

A 2.8 cm tumour has a ratio of 0.933333 and a SAB optimized source position as shown in Table 5.11.

Table 5.11: Predicted source positions for 2.8 cm tumour.

Desired Positions (pixels) (x, y)	**Actual Positions (pixels) (x, y)**	**Euclidian Distance (pixels)**	**Error (mm)**
10,10	9,8	2.24	2.09
12,21	9,20	3.16	2.95
10,31	9,31	1.00	0.93
21,12	21,9	3.00	2.80
21,30	21,29	1.00	0.93
31,10	31,10	0.00	0.00
30,21	30,20	1.00	0.93
31,31	31,31	0.00	0.00

Thus the maximum error for a 2.8 cm tumour is 2.95 mm. This is the most error encountered in the output and is considered the worst case error the of the designed brachytherapy optimization ANN. The resulting hyperdose sleeves for the calculated positions have more that 200% of the dose leaving the tumour whereas the desired positions do not have this level of dose leaving the tumour. All of the output generated with the ANN takes in the order of 100 msec to generate thus it is an extremely fast process.

5.7 Conclusions for ANN

It has been found that an ANN can be used to optimize brachytherapy implants. A few different forms of input were tried in an attempt to find the best form of input for training the ANN. It was found that when the energy function is used for input and output that an ANN with 901 input nodes, 12 internal nodes, and 900 output nodes can achieve an RMS error of 2.03% on a training set and an RMS error of 13.37% on a test set, which corresponds to a maximum source position error of 4 mm. The 4 mm of positional error was calculated assuming that the full 13.37% RMS error results in a shift of the source. However, when the true positions of the predicted sources from the ANN output are checked, they only produce a maximum error of 3 mm. Although a 3 mm error seems large it was generated on the 2.8 cm tumour which has a very large ratio of pixels to mm (the ratio is 0.9333). If a larger ANN is created with more input nodes the ratio could be increased and the error of the 2.8 cm tumour would decrease. Also this was the largest error that was encountered, the average positional error on all of the test sets was only 1.07 mm.

5.8 Chapter Summary

Upon evaluating the output from the various programs developed for this thesis they have been shown to be correct. The SAB software is able to optimize source positions in 2D square tumours up to 3 cm in diameter. The output from the SAB software is then used to train an ANN. Through experimentation and with the use of a cost function it is shown that a BP ANN that uses 901 input nodes, 12 internal nodes, and 900 output nodes is also able to optimize 2D square tumours up to 3 cm in diameter with a worst case error of 3 mm and an average error of 1.07 mm. However, the output from the ANN takes considerably less time to generate, and is measured to take approximately 100 msec as compared to the hours required to generate the SAB output. When the results of this thesis are compared with some of the other work in this area, the benefits that an ANN can offer to this research area are very apparent. The ANN was shown to generate optimized source positions, and it did so 1800 times faster than the work presented in [YuSc96], 900 times faster than [PTR96a], 300 times faster than [YRPZ98], and 180 times faster than the fastest time reported in the literature [MZRB99]. Although this thesis is only the first step in the use of ANNs for brachytherapy optimization it is without a doubt a feasible and logical progression.

CHAPTER VI

CONCLUSIONS AND RECOMMENDATIONS

"The art of drawing conclusions from experiments and observations consists of evaluating probabilities and in estimating whether they are sufficiently great or numerous enough to constitute proofs. This kind of calculation is more complicated and more difficult than it is commonly thought to be..."
- Antoine Laurent Lavoisier (1743-94)
- French chemist, founder of modern chemistry

6.1 Conclusions

This thesis presents the development of a brachytherapy optimization system using an ANN. It is the first attempt at such an optimization scheme for brachytherapy and a number of issues had to be resolved. The thesis consists of three major tasks: the creation of training data using SA, the development of an ANN from the training data, and finally testing the developed ANN to ensure that it is functioning correctly. Three software programs were created for each of the three steps (SAB, MANN, and BowANN) and the design and implementation of these programs is presented.

Training data for the ANN was created using SAB. By evaluating the output from SAB it can be seen that SAB is producing optimized brachytherapy treatments. The output from SAB takes a significant amount of time to generate, demonstrating the need for a faster approach. The output from SAB is then used to train an ANN.

The training of the ANN is accomplished using the MANN software. The MANN software uses the training data created with SAB to develop a single layer BP ANN. We used the MANN software to answer the four questions from Chapter 1. First of all, after trying various forms of input, it was found that the current energy distribution in the tumour is the best form of input for training an ANN. Secondly, it was found that using the desired energy distribution in the tumour as output was the most successful. Thirdly, using an empirical formula it was found that an ANN with 901 inputs, 12 interior nodes, and 900 outputs should perform the best for 3.0 cm square tumours. And finally, this ANN configuration yields an RMS error of 2.03% difference between the correct dose distribution and the predicted dose distribution for the training data indicating that an ANN can learn to optimize brachytherapy implants with relatively little error. The 2.03% difference in dose distribution results in a positional error of 0 mm for the sources on the training data. The ANN was validated using MANN test data, which consists of inputs the ANN has not seen during the training process. Using this data, the ANN achieved an RMS error of 13.37% for the dose distribution prediction, and a maximum positional error of 3 mm which is high for brachytherapy insertions. However, recommendations are made on how to decrease this error in Section 6.2. The average positional error on the test set is only 1.07 mm which is more in the range of what would be considered acceptable for brachytherapy insertions.

When the system developed in this thesis is compared to other work in the field it is apparent that significant improvement was achieved. The speed of the ANN approach is more than 180 times faster than the next fasted method presented in the literature [MZRB99]. Although the ANN is currently limited to 2D tumours, extending the concept to handle 3D would not decrease the output speed too drastically, and therefore it is clear that this new method is far superior to the other methods in terms of speed. The second major improvement that results from this new approach is the ability of the ANN to place sources anywhere within the implant volume. All of the current literature restricts the sources to regular array geometries [PTR96b][Slob92][YRPZ98][YuSc96].

The system developed in this thesis produces very good results for the cases for which it was designed. However, there are limitations. First of all, the input is limited to 2D square tumours up to 3 cm in width. Clearly the system has to be expanded to handle any shape of tumour and operate in 3D before it would be clinically viable. In the current design, the ANN can also produce false positives, which are misleading. These false positives occur when the ANN is used to produce output for very extreme inputs (in which sources have been placed at unexpected locations). All of the limitations identified however can be resolved with additional research and development of the technique.

It can be deduced from this research that there is a very large area to be investigated in the use of ANNs for brachytherapy optimization. Hopefully as a result of this thesis a new path has been created in the optimization of brachytherapy that will see many other researchers investigating the possibilities. One can foresee a day in which all cancer patients treated with brachytherapy will have custom treatments created and updated as the procedure is carried out resulting in a tremendous increase in the quality of life for those patients.

6.2 Recommendations

In order to over come the limitations of the research which have been identified the following recommendations are suggested for further research on this topic:

1) In order to decrease the maximum error achieved in the results of this thesis, there are two possible approaches. First of all, the size of the ANN could be increased. The positional error of sources increases as the ratio of the tumour decreases. Therefore, if we had in the order of three times as many input and output nodes, it should result in maximum positional errors for the larger tumour sizes similar to the positional errors currently found at smaller sizes (approximately 1.0 mm). Secondly, we could approach the problem using a different ANN structure that is more capable of complex problems, and able to learn new cases as they appear. The Probabilistic Neural Network (PNN) is a very likely candidate. For information on the PNN, consult [Mast93].

2) Larger training sets for the ANN, consisting of tumours of all shapes and sizes. will increase the possibility of this technique being used clinically. This will result in the creation of an ANN which can produce output for a wider range of inputs.

3) Once the ANN is able to produce correct results for most common shapes and sizes of tumours, it should be extended to handle 3D input. This would then make this process on par with the current optimization techniques which use SA and GA for optimization. However, with the speed advantage of ANNs it would be more advantages than the current techniques.

4) Finally, the most difficult issue in optimizing brachytherapy treatments can be introduced, variable source dwell times which will improve the results even further. This would still be accomplished by training the ANN with SA data, however the SA would now have variable strength sources in the results.

6.3 Contributions

This thesis has made the following contributions:

1) Identifies a method for applying advanced artificial intelligence techniques to brachytherapy which has not been done before. The work done in this thesis identifies that by using the current energy distribution in the tumour it is possible to train an ANN for brachytherapy optimization. It the work was extended to include a representation of the tumour (as opposed to using a ratio) it is possible that this technique would be clinically viable;

2) Applies ANNs to an new area which has never been tried before. The more wide the range of applications that ANNs are applied to, the more we learn of their abilities. It also adds to the general knowledge base of applications for which ANNs are a viable option; and

3) Has produced optimized brachytherapy treatments 180 times faster than current best technique. Even once the work is expanded to include variable tumour shapes and sizes, the reduction in speed is not going to be significant. Thus this technique will still be considerable faster than any of the non-deterministic approaches.

REFERENCES

[AaKo89] E. Aarts and J. Korst, *Simulated annealing and Boltzmann machines : a stochastic approach to combinatorial optimization and neural computing.* NY, NY: John Wiley & Sons, Inc., 1998, 272 pp.

[ASEF97] R. Alfredo, C. Siochi, H. R. Elson, A. E. Foster, and M. A. Lamba, "A self-convolution backprojection algorithm for optimizing dose districutions of I-125 prostate implants," *Med. Phys.*, vol. 24, no. 2, pp. 241-249, 1997.

[AEAA97] Y. Anacak, M. Esassolak, A. Aydm, A. Aras, I. Olacak, and A. Haydaroglu, "Effect of geometric optimization on the treatment volumes and the dose homogeneity of biplane interstitial brachytherapy implants," *Radiotherapy and Oncology*, vol. 45, no. 1, pp. 71-76, 1997.

[Ande86] L. L. Anderson, "A natural volume-dose histogram for brachytherapy," *Med. Phys.*, vol. 13, no. 6, pp. 898-903, 1986.

[BSSL88] B. Bauer-Kirpes, V. Strun, W. Schlegel, and W. J. Lorenz, "Computerized optimization of 125-I implants in brain tumors," *Int. J. Radiation Oncology Biol. Phys.*, vol. 14, no. 5, pp. 1013-1023, 1988.

[BOCS90] G. C. Bentel, J. R. Oleson, D. Clarke-Pearson, J. T. Soper, and G. S. Montana, "Transperineal Templates for Brachytherapy Treatment of Pelvic Malignancies - A Comparison of Standard and Custom Templates," *Int. J. Radiation Oncology Biol. Phys.*, vol. 19, pp. 751-758, 1990.

[Cern85] V. Cerny, "Thermodynamical Approach to the Traveling Salesman Problem: An Efficient Simulation Algorithm", *J. Opt. Theory Appl.*, vol. 45, no. 1, pp. 41-51, 1985.

[ChMu98] V. Cherkassy and F. Mulier, *Learning From Data - Concepts, Theory, and Methods.* NY, NY: John Wiley & Sons, Inc., 1998.

[ChoJ00] J. M. Cho, "Chromosome Classification Using Backpropagation Neural Networks," *IEEE Engineering in Medicine and Biology*, vol. January/February, pp. 28-33, 2000.

[CPPR99] C. Citterio, A. Pelagotti, V. Piuri, and L. Rocca, "Function Approximation

- A Fast Convergence Neural Approach Based on Spectral Analysis," *IEEE Transaction on Neural Networks*, vol. 10, no. 4, pp. 725-740, 1999.

[Deni01] A. Denis. *Fast and Reliable Validation System for Printed Documents*, M.Sc. Thesis, University of Manitoba, November 2001, 321 pp.

[Dict00] Lexico dictionary. From http://www.dictionary.com (available as of January 1999).

[Dutr88] A. Dutreix, "Can we Compare Systems for Interstitial Therapy," *Radiotherapy and Oncology*, vol. 13, pp. 127-135, 1988.

[Edmu90] G. K. Edmundson, Chapter 16: Geometry Based Optimization For Stepping Source Implants, In: *Brachytherapy HDR and LDR*, eds. A. A. Martinez, C. G. Orton, and R. F. Mould. Columbia, MD: Nucletron Corporation, 1990, pp. 184-192.

[ERTB93] G. K. Edmundson, N. R. Rizzo, M. Teahan, D. Brabbins, F. A. Vivini, and A. A. Martinez, "Concurent treatment planning for outpatient high dose rate prostate template implants," *Int. J. Radiation Oncology Biol. Phys.*, vol. 27, no. 5, pp. 1215-1223, 1993.

[EdYM95] G. K. Edmundson, D. Yan, and A. A. Martinez, "Intraoperative optimization of needle placement and dwell times for conformal prostate brachytherapy," *Int. J. Radiation Oncology Biol. Phys.,* vol. 33, no. 5, pp. 1257-1263, 1995.

[EJMR98] A. Eisbruch, C. M. Johnston, M. K. Martel, J. M. Robertson, K. R. Reynolds, L. H. Marsh, and J. A. Roberts, "Customized Gynecologic Interstitial Implants: CT-Based Planning, Dose Evaluation, and Optimization Aided by Laparotomy," *Int. J. Radiation Oncology Biol. Phys.,* vol. 40, no. 5, pp. 1087-1093, 1998.

[ErAG96] B. Erickson, K. Albano, and M. Gillin, "CT-Guided Interstitial Implantation of Gynecologic Malignancies," *Int. J. Radiation Oncology Biol. Phys.,* vol. 36, no. 3, pp. 699-709, 1996.

[EzLu95] G. A. Ezzell and R. W. Luthmann, Chapter 30: Clinical Implementation of Dwell Time Optimization Techniques for Singel Stepping-Source Remote Application. In: *Brachytherapy Physics*, eds. G. A. Ezzell and R. W. Luthmann. Madison, WI: MP publishing, 1995.pp. 617-639.

[FPSF96] B. Farrus, F. Pons, A. Sanchez-Reyes, F. Ferrer, A. Rovirosa, and A. Biete, "Quality Assurance of Interstitial Brachytherapy techniques in lip cancer: Comparison of Actual Performance with the Paris System Recomendations," *Radiotherapy and Oncology*, vol. 38, pp. 145-151, 1996.

[Fosn98] R. Fosner, *OpenGL Programming for Windows 95 and Windows NT*. Reading, Mass: Addison-Wesley Developers Press, 1998, 230 pp.

[FHBS96] P. Frizt, F. W. Hensley, C. Berns, P. Schraube, and M. Wannenmacher, "First experiences with superfractionated skin irradiations using large afterloading molds," *Int. J. Radiation Oncology Biol. Phys.*, vol. 36, no. 1, pp. 147-157, 1996.

[FuIs00] Y. Fukuoka and A. Ishida, "Chronic Stress Evaluation Using Neural Networks," *IEEE Engineering in Medicine and Biology*, vol. January/February, pp. 34-38, 2000.

[GKWC84] M. Gillin, R. W. Kline, J. F. Wilson, and J. D. Cox, "Single and Double Plane Implants: A Comparison of the Manchester System with the Paris System," *Int. J. Radiation Oncology Biol. Phys.*, vol. 10, pp. 921-925, 1984.

[Gins97] M. Ginsberg, *Essentials of Artificial Intelligence*. San Francisco, CA: Morgan Kauffman Publishers Inc., 1997, 430 pp.

[Godd88] T. J. Godden, *Physical Aspects of Brachytherapy*. Bristol, England: IOP, 1988, 256 pp.

[GAHL94] P. C. Griffin, P. A. Amin, P. Hughes, A. M. Levine, W. W. Sewchand, and O. M. Salazar, "Pelvic Mass: CT-guided Interstitial Catheter Implantation with High Dose Rate Remote Afterloader," *Radiology*, vol. 191, no. 2, pp. 581-583, 1994.

[HaBr92] E. J. Hall and D. J. Brenner, "The Dose-Rate Effect in Interstitial Brachytherapy: A Controversy resolved," *The British Journal of Radiology*, vol. 65, pp. 242-247, 1992.

[HMSR91] T. Holmes, T. R. Mackie, D. Simpkin, and P. Reckwerdt, "A Unified Approach to the Optimization of Brachytherapy and External Beam Dosimetry," *Int. J. Radiation Oncology Biol. Phys.*, vol. 20, pp. 859-873, 1991.

[JeKo00] Jeri Kostyra, "Cancer treatments tabulated data," Manitoba Cancer Registry, Winnipeg, MB, Canada. (Private communications).

[Kasa96] N. K. Kasabov, *Foundations of Neural Networks, Fuzzy Systems, and Knowledge Engineering*. Cambridge, Mass: The MIT Press, 1996, 550 pp.

[KiGV83] S. Kirkpatrick, C. D. Gelatt Jr., M. P. Vecchi, "Optimization by Simulated Annealing", *Science*, vol. 220, no. 4598, pp. 671-680, 1983.

[KVDN94] I. K. K. Kolkman-Deurloo, A. G. Viser, N. Driver, C. G. J. H. Niel, and P. C. Levendag, "Optimization of interstitial colume implants," *Radiotherapy and Oncology*, vol. 31, pp. 229-239, 1994.

[KuAn99] M. A. Kupinski and M. A. Anastasio, "Multiobjective Genetic Optimization of Diagnostic Classifiers with Implications for Generating Receiver Operating Charactersitics Curves," *IEEE Transactions on Medical Imaging*, vol. 18, no. 8, pp. 675-685, 1999.

[KKOC83] D. K. Kwan, A. R. Kagan, A. J. Olch, P. Y. M. Chan, B. L. Hintz, and M. Wollin, "Single- and double-plane iridium-192 interstitial implants: Implantation guidelines and dosimetry," *Med. Phys.*, vol. 10, no. 4, pp. 456-461, 1983.

[LaBZ99] M. Lahanas, D. Baltas, and N. Zamboglou, "Anatomy-Based Three-Dimensional Dose Optimization in Brachytherapy Using Multiobjective Genetic Algorithms," *Med. Phys.*, vol. 26, no. 9, pp. 1904-1918, 2000.

[LaAH97] J. Laitinen, J. Alakuijala, and H. Helminen, "Maximum dose projection: A new 3D dose visualization tool for brachytherapy," *XIIth ICCR May 27-30*, vol. 1, no. 1, pp. 137-139, 1997.

[LaTh95] K. A. Langmack and S. J. Thomas, "The Application of Dose-Volume Histograms to the Paris and Manchester Systems of Brachytherapy Dosimetry," *The British Journal of Radiology*, vol. 68, pp. 42-48, 1995.

[LeDM97] D. F. Leotta, P. R. Detmer, and R. W. Martin, "Performance of a miniature magnetic position sensor for three-dimensional ultrasound imaging," *Ultrasound in Med. & Biol.*, vol. 23, no. 4, pp. 597-609, 1997.

[LiTa00] C. H. Li and P. K. S. Tam, "A Global Energy Approach to Facet Model and its Minimization Using Weighted Least-Squares Algorithm," *Pattern Rec-*

ognition, vol. 33, pp. 281-293, 2000.

[Mast93] T. Masters, *Practical Neural Network Recipes in C++*. San Diego, CA: Academic Press, 1993, 493 pp.

[McRu88] J. L. McClelland and D. E. Rumelhart, *Parallel Distributed Processing - Exploration in the Microstructure of Cognition - Volume 2: Psychological and Biological Models*. Cambridge, Mass: The MIT Press, 1988, 611 pp.

[MeMR97] K. Mehrotra, C. K. Mohan, and S. Ranka, *Elements of Artificial Neural Networks*. Cambridge, Mass. The MIT Press, 1997, 344 pp.

[MiPa69] M.L. Minsky and S.A. Papert. *Perceptrons: An introduction to computational geometry*. Cambridge, Mass. The MIT Press, 1969.

[MZRB99] E. M. Messing, J. B. Zhang, D. J. Rubens, R. A. Brasacchio, J. G. Strang, A. Soni, M. C. Schell, P. G. Okunieff, and Y. Yu, "Intraoperative Optimized Inverse Planning for Prostate Brachytherapy: Early Experience," *Int. J. Radiation Oncology Biol. Phys.,* vol. 44, no. 4, pp. 801-808, 1999.

[MJBK99] S. Miller, C. Jeffrey, J. Bews, and W. Kinsner, "Advances in the virtual reality interstitial brachytherapy system," Edmonton, Alberta: *IEEE Conference Proceedings, CCECE'99*, pp. 349-354, May 9-12, 1999.

[MBBM98] S. Miller, A. Berndt, J. Bews, B. McCurdy and W. Kinsner, "An implementation of a virtual reality interstitial brachytherapy system," Warterloo, Ontario: *IEEE Conference Proceedings, CCECE'98*, pp. 870-873, May 24-28, 1998.

[NCIC02] National Cancer Institute of Canada: *Canadian Cancer Statistics 2002*, Toronto, Canada, 2002.

[Padb99] M. Padberg, *Linear Optimization and Extensions*. Mercedes-Druck, Berlin: Springer, 19pp, 501pp.

[PPBK89] J. M. Paul, P. C. Philip, R. W. Brandenburg, and R. F. Koch, "Comparison Between Continuous and Discrete Sources in the Paris System of Implants," *Med. Phys.,* vol. 16, no. 3, pp. 414-424, 1989.

[PKKB00] S. Pavlopoulos, E. Kyriacou, D. Koutsouris, K. Blekas, A. Stafylopatis, and P. Zoumpoulis, "Fuzzy Neural Network - Based Texture Analysis of

Ultrasonic Images," *IEEE Engineering in Medicine and Biology*, vol. January/February, pp. 39-47, 2000.

[PiEl98] S. Pierre and A. Elgibaoui, "Dimensioning computer network using tabu search," Warterloo, Ontario: *IEEE Conference Proceedings, CCECE'98*, pp. 834-838, May 24-28, 1998.

[PiGr84] S. Pistorius and W. A. Groenewald. *The determination of source-position times in intracavitary radiotherapy*, 1984. (UnPub)

[PTR96a] J. Pouliot, D. Tremblay, J. Roy, and S. Filice, "Optimization of permanent I-125 prostate implants using fast simulated annealing," *Int. J. Radiation Oncology Biol. Phys.,* vol. 36, no. 3, pp. 711-720, 1996.

[PTR96b] J. Pouliot, D. Tremblay, J. Roy, and S. Filice, "Automatic optimization of permanent I-125 prostate implants," *COMP96*, vol. 1, no. 1, 1996.

[PTRV97] J. Pouliot, D. Tremblay, J. Roy, E. Vigneault, and L. Girouard, "Prostate cancer permanent implants, role of radio-opaque markers and epid," *CCPM97*, vol. 1, no. 1, 1997.

[Rang00] A. Rangaranjan, "Self-Annealing and Self-Annihilation: Unifying Deterministic Annealing and Relaxation Labeling," *Pattern Recognition*, vol. 33, pp. 635-649, 2000.

[RaRa95] V. B. Rao and H. V. Rao, *Neural Networks and Fuzzy Logic.* NY, NY: MIS Press, 1995, 551 pp.

[RWCA91] J. N. Roy, K. E. Wallner, S. Chiu-Tsao, L. L. Anderson, and C. C. Ling, "CT-Based optimized planning for transperineal prostate implants with costomized template," *Int. J. Radiation Oncology Biol. Phys.,* vol. 21, no. 2, pp. 483-489, 1991.

[RuMc88] D. E. Rumelhart and J. L. McClelland, *Parallel Distributed Processing - Exploration in the Microstructure of Cognition - Volume 1: Foundations.* Cambridge, Mass: The MIT Press, 1988, 547 pp.

[SaWu98] C. Saw and A. Wu, "Evaluation of the Substitution of Ir192 Seed Ribbons for Wires in Paris System using dose Nonuniformity Ratio," *Int. J. Radiation Oncology Biol. Phys.,* vol. 25, pp. 551-556, 1998.

[STPS00] F. Schnorrenberg, N. Tsapatsoulis, C. S. Pattichis, C. N. Schizas, S. Kollias, M. Vassiliou, A. Adamou, and K. Kyriacou, "Improved Detection of Breast Cancer Nuclei Using Modular Neural Networks," *IEEE Engineering in Medicine and Biology*, vol. January/February, pp. 48-62, 2000.

[Slob92] R. Sloboda, "Optimization of brachytherapy dose distributions by simulated annealing," *Med. Phys.,* vol. 19, no. 4, pp. 955-964, 1992.

[StTh97] J. A. Stitt and B. R. Thomadsen, "Innovations and Advances in Brachytherapy," *Seminars in Oncology*, vol. 24, no. 6, pp. 696-706, 1997.

[TKTS98] S. Tang, C. Kwoh, M. Teo, N. Sing, and K. Ling, "Augmented reality systems for medical applications," *IEEE Engineering in Medicine and Biology*, vol. May/June, pp. 49-58, 1998.

[VaDe90] R. Van der Laarse and R. W. De Boer. Computerized high dose rate brachytherapy treatment planning. In: *Brachytherapy HDR and LDR*, eds. A. A. Martinez, C. G. Orton, and R. F. Mould. Columbia, MD: Nucletron Corporation, 1990.pp. 169-183.

[VJHE98] F. A. Vicini, D. A. Jaffray, E. M. Horwitz, G. K. Edmundson, D. A. DeBiose, V. R. Kini, and A. A. Martinez, "Implementation of 3D-Virtual Brachytherapy in the Management of Breast Cancer: A description of a new Method of Interstitial Brachytherapy," *Int. J. Radiation Oncology Biol. Phys.,* vol. 40, no. 3, pp. 629-635, 1998.

[WaPr99] C. Wang and J. C. Principe, "Training Neural Networks with Additive Noise in the Desired Signal," *IEEE Transaction on Neural Networks*, vol. 10, no. 6, pp. 1511-1517, 1999.

[WaMS00] Z. Wang, M. T. Manry, and J. L. Schiano, "LMS Learning Algorithms: Misconceptions and New Results on Convergence," *IEEE Transaction on Neural Networks*, vol. 11, no. 1, pp. 47-56, 2000.

[Wass89] P. D. Wasserman, *Neural Computing - Theory and Practice*. NY, NY: Van Nostrand Reinhold, 1989, 303 pp.

[WaAn97] Y. Watanabe and L. L. Anderson, "A system for nonradiographic source localization and real-time planning of intraoperative high dose rate brachytherapy," *Med. Phys.,* vol. 24, no. 12, pp. 2014-2023, 1997.

[WCFV98] C. De Wagter, C. O. Colle, L. G. Fortan, B. B. Van Duyse, D. L. Van den Berge, and W. J. De Neve, "3D Conformal Intensity-Modulated Radiotherapy Planning: Interactive Optimization by Constrained Matrix Inversion," *Radiotherapy and Oncology*, vol. 47, pp. 69-76, 1998.

[WeKn97] R. Wehrmann and P. Kneschaurek, "Analytic and Numerical Solutions for an inverse problem in brachytherapy," *XIIth ICCR May 27-30*, vol. 1, no. 1, pp. 127-130, 1997.

[WeKu91] S. M. Weiss and C. A. Kulikowski, *Computer Systems That Learn*. San Mateo, CA: Morgan Kaufmann Publishers, Inc. 1991, 223 pp.

[WuUS88] A. Wu, K. Ulin, and E. S. Sternick, "A Dose Homogeneity Index for Evaluating 192Ir Interstitial Breast Implants," *Med. Phys.,* vol. 15, no. 1, pp. 104-107, 1988.

[YRPZ98] G. Yang, L. E. Reinstein, S. Pai, and Z. Xu, "A new genetic algorithm technique in optimization of permanent 125I prostate implants," *Med. Phys.,* vol. 25, no. 12, pp. 2308-2315, 1998

[YuSc96] Y. Yu and M. C. Schell, "A genetic algorithm for the optimization of prostate implants," *Med. Phys.,* vol. 23, no. 12, pp. 2085-2091, 1996..

APPENDIX A
ADDITIONAL SAB RESULTS

A.1 Square Tumour 0.8 cm in Width

The 0.8 cm tumour shown in Fig. A.1 was used as input to the SAB software. The configuration file shown in Fig. A.2 was used to control the flow of the SAB software as discussed in Section 4.3.1.2. This trial produced a cost iteration plot as shown in Fig. A.3. A cooling profile shown in Fig. A.4 was utilized. The output from the SAB software is as shown in Fig. A.5. The hyperdose sleeve for the solution shown in Fig. A.5, is shown in Fig. A.6. Since the cost function for SAB is essentially comprised of two factors, the homogeneity of the dose in a tumour and the size of the hyperdose sleeves, it is easy to predict the output for a small tumour (0.8 cm). The results shown in Fig. A.5 are exactly as expected. Since the tumour size is smaller than the maximum hyperdose sleeve size, a single source could be placed anywhere in the tumour and meet that requirement in the SA cost function. However, because of the second component of the SA cost function we would expect that a homogeneous dose in the tumour would be more desirable. As the SAB software attempts to find the best solution with the minimum number of sources possible, a single source at the center is exactly what we expect. The output from the SAB algorithm for a tumour size of 0.8 cm is therefore shown to be correct. The final cost function value for this source distribution was 2.290165.

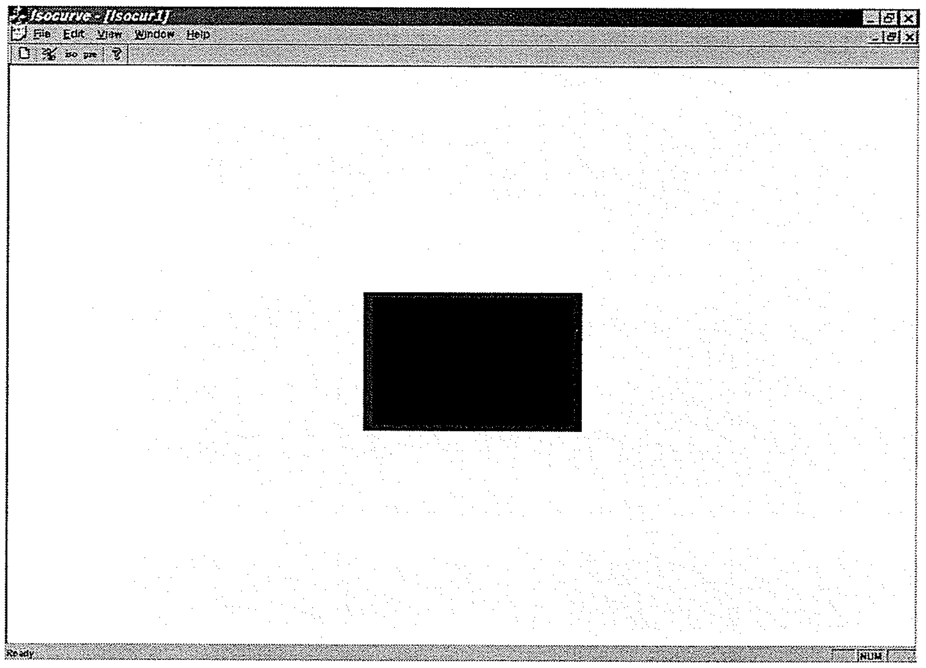
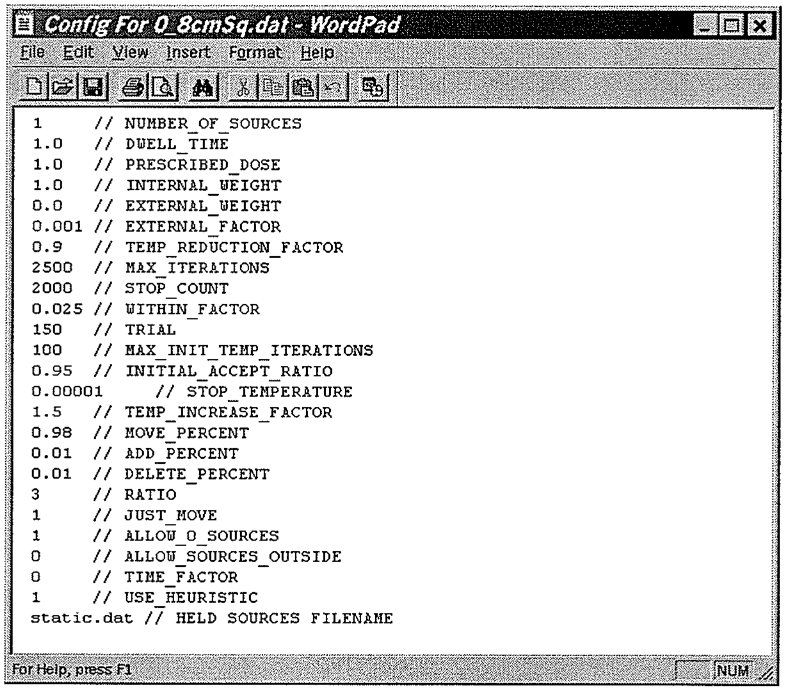


Fig. A.1 0.8 cm tumour to be optimized with SAB.


Config For 0_8cmSq.dat - WordPad
File Edit View Insert Format Help
For Help, press F1
NUM

```
1      // NUMBER_OF_SOURCES
1.0    // DWELL_TIME
1.0    // PRESCRIBED_DOSE
1.0    // INTERNAL_WEIGHT
0.0    // EXTERNAL_WEIGHT
0.001 // EXTERNAL_FACTOR
0.9    // TEMP_REDUCTION_FACTOR
2500  // MAX_ITERATIONS
2000  // STOP_COUNT
0.025 // WITHIN_FACTOR
150    // TRIAL
100    // MAX_INIT_TEMP_ITERATIONS
0.95  // INITIAL_ACCEPT_RATIO
0.00001      // STOP_TEMPERATURE
1.5    // TEMP_INCREASE_FACTOR
0.98  // MOVE_PERCENT
0.01  // ADD_PERCENT
0.01  // DELETE_PERCENT
3      // RATIO
1      // JUST_MOVE
1      // ALLOW_0_SOURCES
0      // ALLOW_SOURCES_OUTSIDE
0      // TIME_FACTOR
1      // USE_HEURISTIC
static.dat // HELD SOURCES FILENAME
```

Fig. A.2 Configuration file for 0.8 cm tumour.

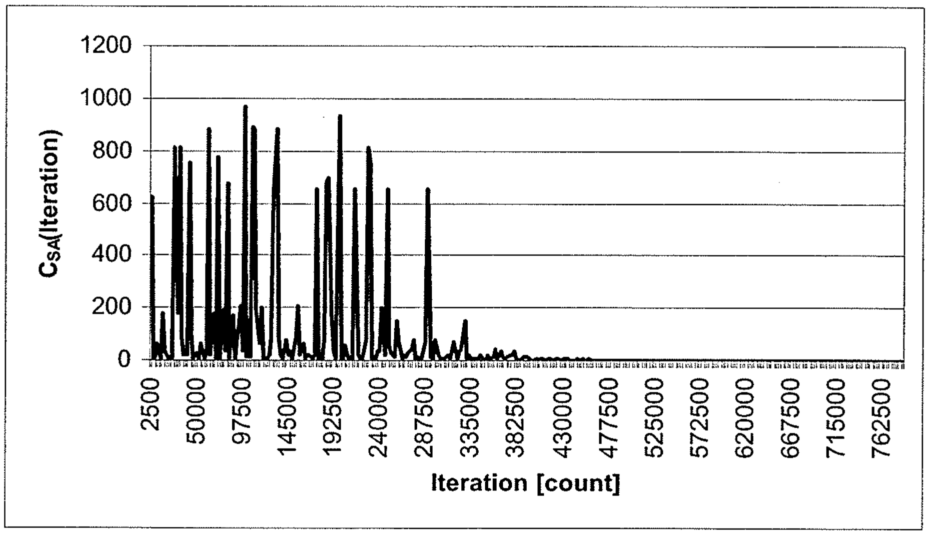


Fig. A.3 Cost plot for 0.8 cm tumour.

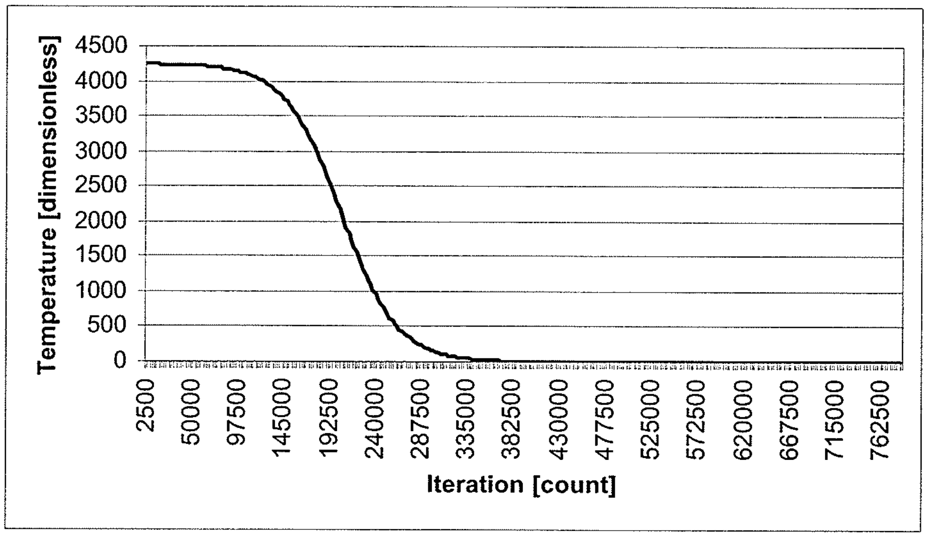


Fig. A.4 Cooling profile for 0.8 cm tumour.

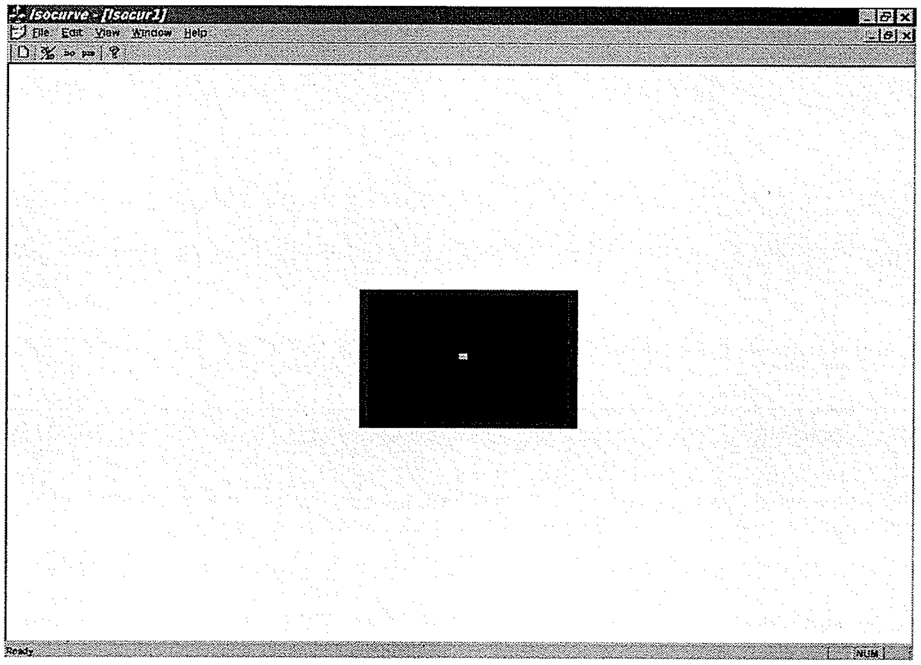


Fig. A.5 Optimized source position for 0.8 cm tumour.

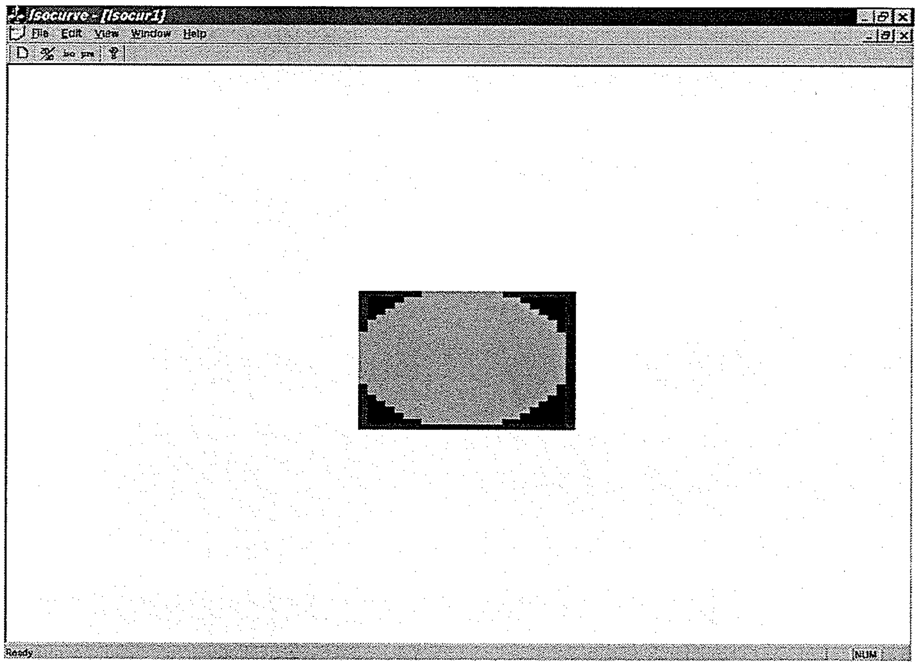


Fig. A.6 Hyperdose sleeve for 0.8 cm solution.

A.2 Square Tumour 1.2 cm in Width

The 1.2 cm tumour shown in Fig. A.7 was used as input to the SAB software. The configuration file shown in Fig. A.8 was used to control the flow of the SAB software as discussed in Section 4.3.1.2. This trial produced a cost iteration plot as shown in Fig. A.9. A cooling profile shown in Fig. A.10 was utilized. The output from the SAB software is as shown in Fig. A.11. The hyperdose sleeve for the solution shown in Fig. A.11, is shown in Fig. A.12. The size of the hyperdose sleeve is obviously larger than the desired threshold of 1.0 cm (since it spans the entire tumour which is 1.2 cm) however, the SAB software ran until the maximum number of sources was reached and this was the best solution that was obtained. Since the solution for 1.5 cm has four sources we would expect that there should be no more than four sources for the 1.2 cm tumour. When we inspect the output from two, three and four sources, the sources are all bunched at the middle (not a homogeneous distribution). This is due to the fact that entirely too much dose is leaving the tumour. Since we are looking for the minimum number of sources that produce an acceptable output, the solution for one source is better than the others. One source has a cost of 2.365548.

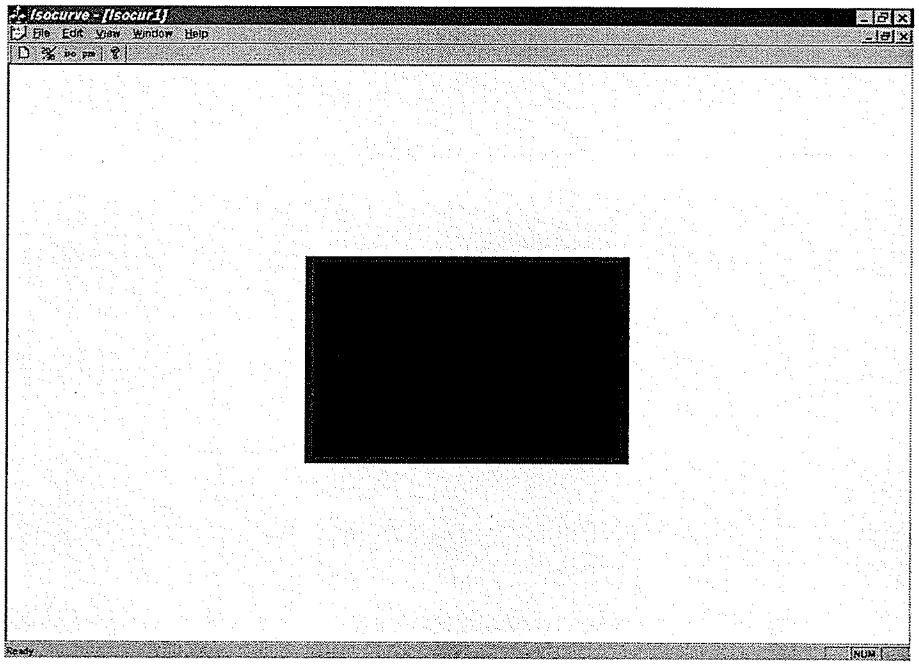
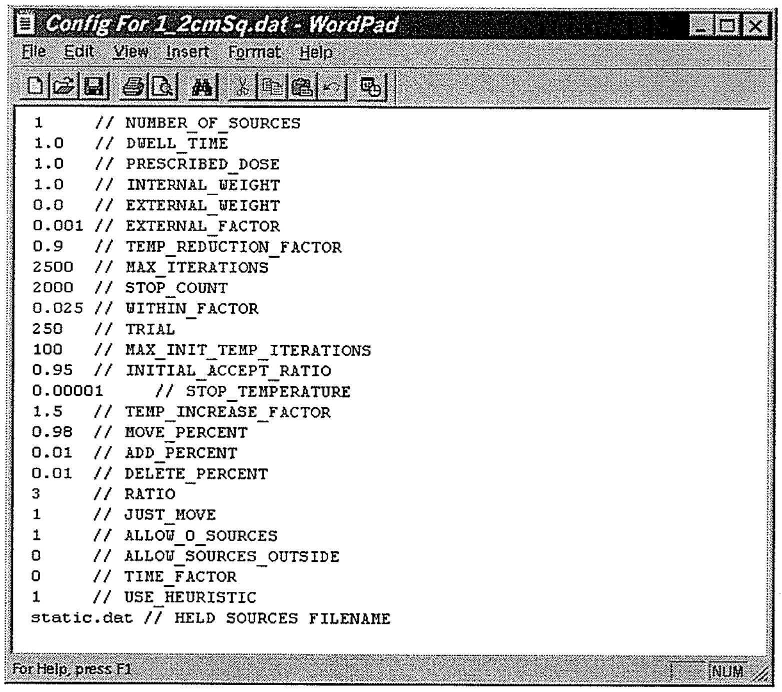


Fig. A.7 1.2 cm tumour to be optimized with SAB.


Config For 1_2cmSq.dat - WordPad
File Edit View Insert Format Help
For Help, press F1
NUM

```
1      // NUMBER_OF_SOURCES
1.0    // DWELL_TIME
1.0    // PRESCRIBED_DOSE
1.0    // INTERNAL_WEIGHT
0.0    // EXTERNAL_WEIGHT
0.001 // EXTERNAL_FACTOR
0.9    // TEMP_REDUCTION_FACTOR
2500  // MAX_ITERATIONS
2000  // STOP_COUNT
0.025 // WITHIN_FACTOR
250    // TRIAL
100    // MAX_INIT_TEMP_ITERATIONS
0.95  // INITIAL_ACCEPT_RATIO
0.00001      // STOP_TEMPERATURE
1.5    // TEMP_INCREASE_FACTOR
0.98  // MOVE_PERCENT
0.01  // ADD_PERCENT
0.01  // DELETE_PERCENT
3      // RATIO
1      // JUST_MOVE
1      // ALLOW_0_SOURCES
0      // ALLOW_SOURCES_OUTSIDE
0      // TIME_FACTOR
1      // USE_HEURISTIC
static.dat // HELD SOURCES FILENAME
```

Fig. A.8 Configuration file for 1.2 cm tumour.

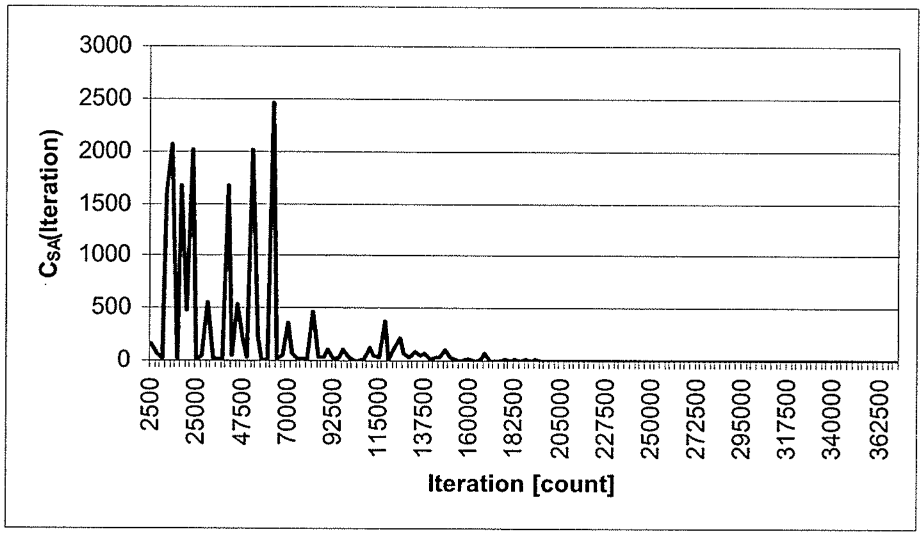


Fig. A.9 Cost plot for 1.2 cm tumour.

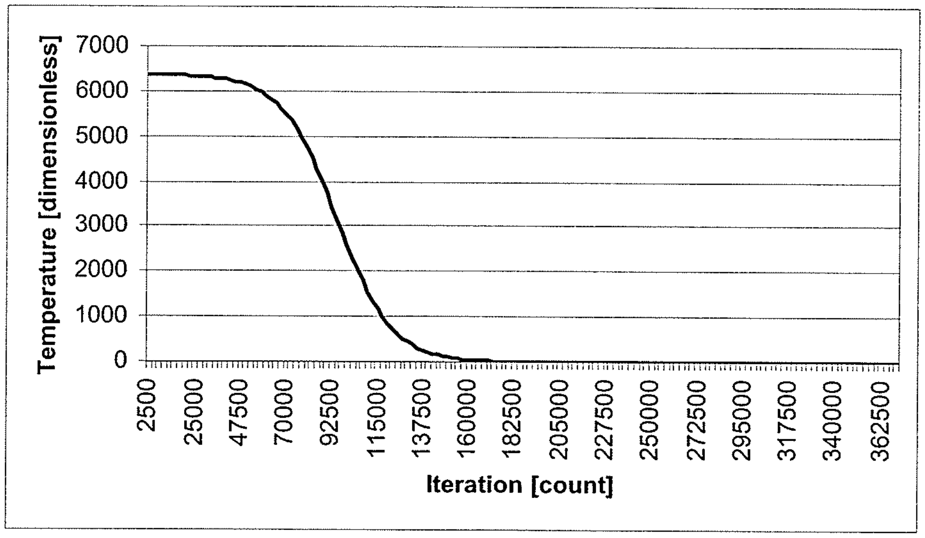


Fig. A.10 Cooling profile for 1.2 cm tumour.

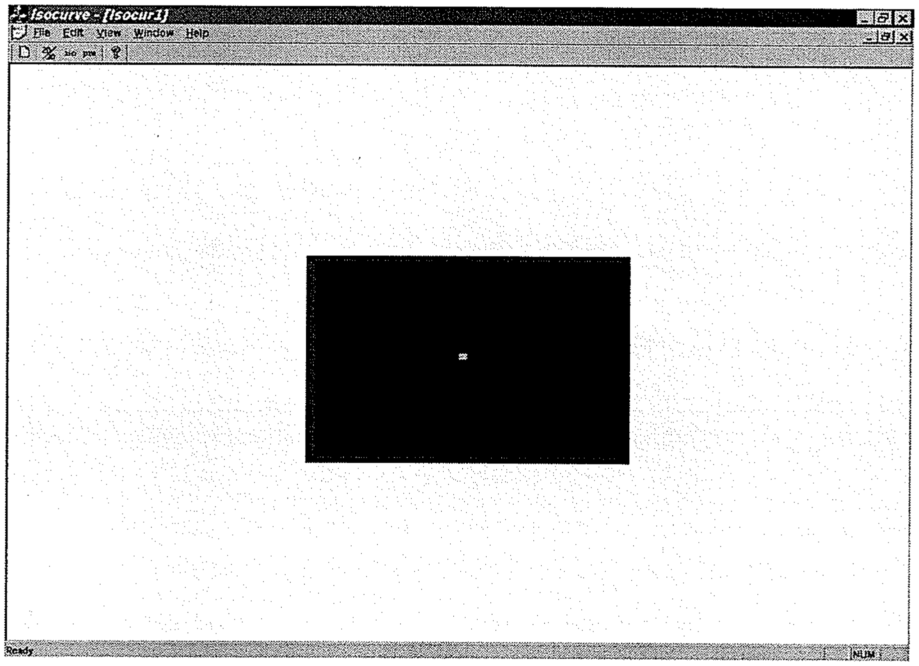


Fig. A.11 Optimized source positions for 1.2 cm tumour.

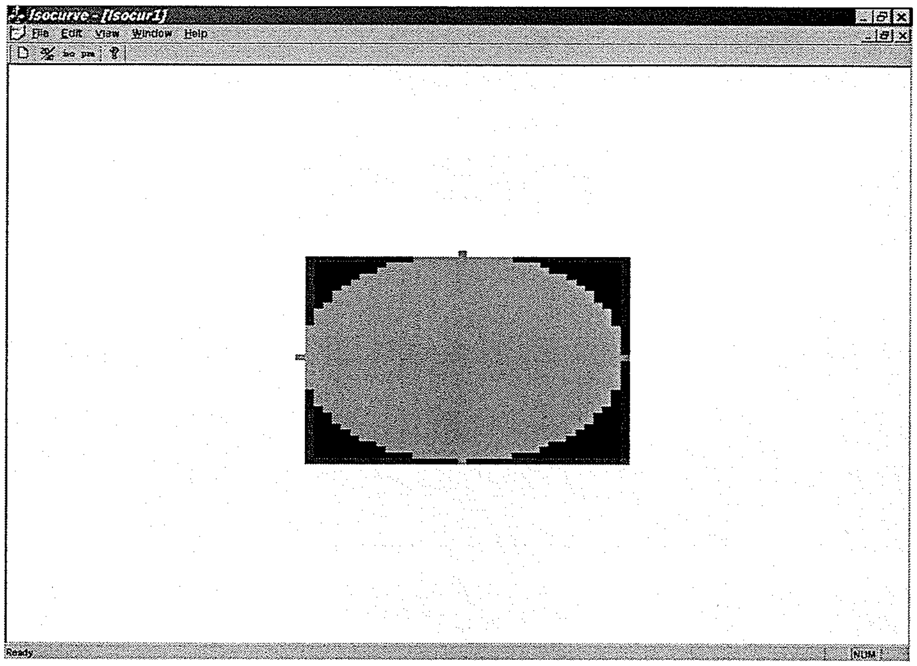


Fig. A.12 Hyperdose sleeve for 1.2 cm solution.

A.3 Square Tumour 1.8 cm in Width

The 1.8 cm tumour shown in Fig. A.13 was used as input to the SAB software. The configuration file shown in Fig. A.14 was used to control the flow of the SAB software as discussed in Section 4.3.1.2. This trial produced a cost iteration plot as shown in Fig. A.15. A cooling profile shown in Fig. A.16 was utilized. The output from the SAB software is as shown in Fig. A.17. The hyperdose sleeve for the solution shown in Fig. A.17, is shown in Fig. A.18. The hyperdose sleeves are measured to be 0.8 cm well within our maximum allowed of 1.0 cm. The four sources are also evenly distributed within the tumour meeting the other requirements of the SAB cost function. The output from the SAB algorithm for a tumour size of 1.8 cm is therefore shown to be correct. The final cost function value for this source distribution was 1.363699.

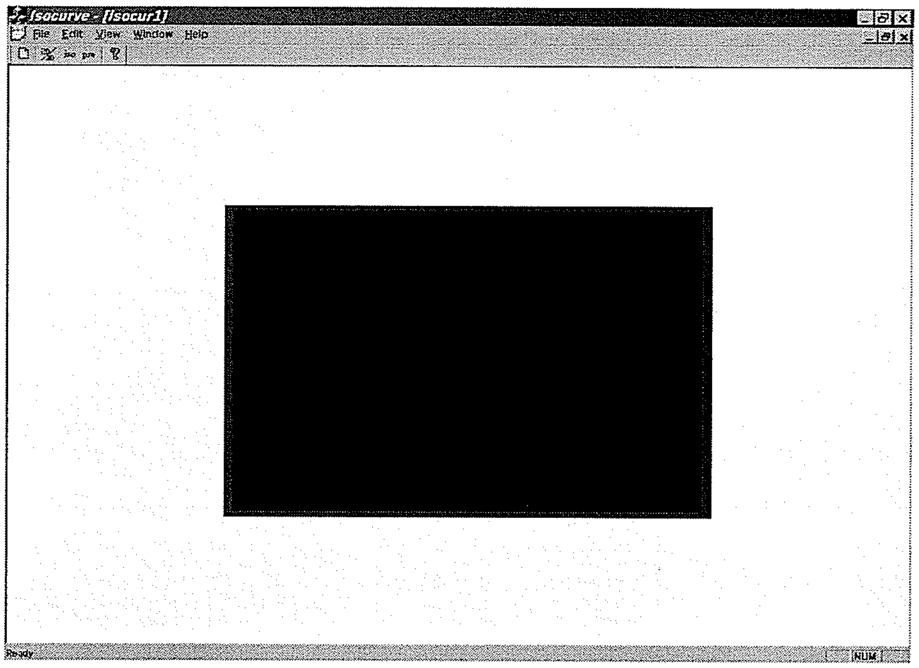
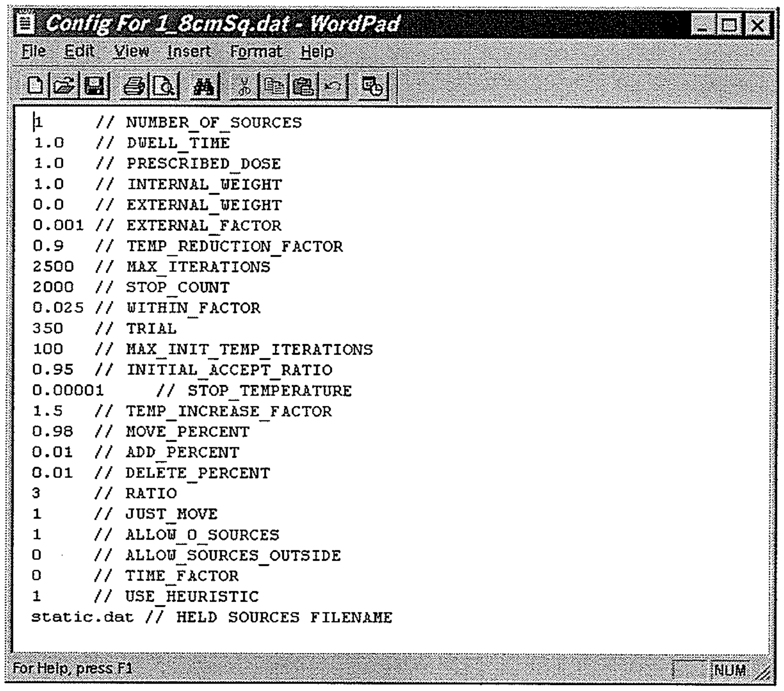


Fig. A.13 1.8 cm tumour to be optimized with SAB.


Config For 1_8cmSq.dat - WordPad
File Edit View Insert Format Help
For Help, press F1
NUM

```
1      // NUMBER_OF_SOURCES
1.0    // DWELL_TIME
1.0    // PRESCRIBED_DOSE
1.0    // INTERNAL_WEIGHT
0.0    // EXTERNAL_WEIGHT
0.001 // EXTERNAL_FACTOR
0.9    // TEMP_REDUCTION_FACTOR
2500  // MAX_ITERATIONS
2000  // STOP_COUNT
0.025 // WITHIN_FACTOR
350    // TRIAL
100    // MAX_INIT_TEMP_ITERATIONS
0.95  // INITIAL_ACCEPT_RATIO
0.00001     // STOP_TEMPERATURE
1.5    // TEMP_INCREASE_FACTOR
0.98  // MOVE_PERCENT
0.01  // ADD_PERCENT
0.01  // DELETE_PERCENT
3      // RATIO
1      // JUST_MOVE
1      // ALLOW_0_SOURCES
0      // ALLOW_SOURCES_OUTSIDE
0      // TIME_FACTOR
1      // USE_HEURISTIC
static.dat // HELD SOURCES FILENAME
```

Fig. A.14 Configuration file for 1.8 cm tumour.

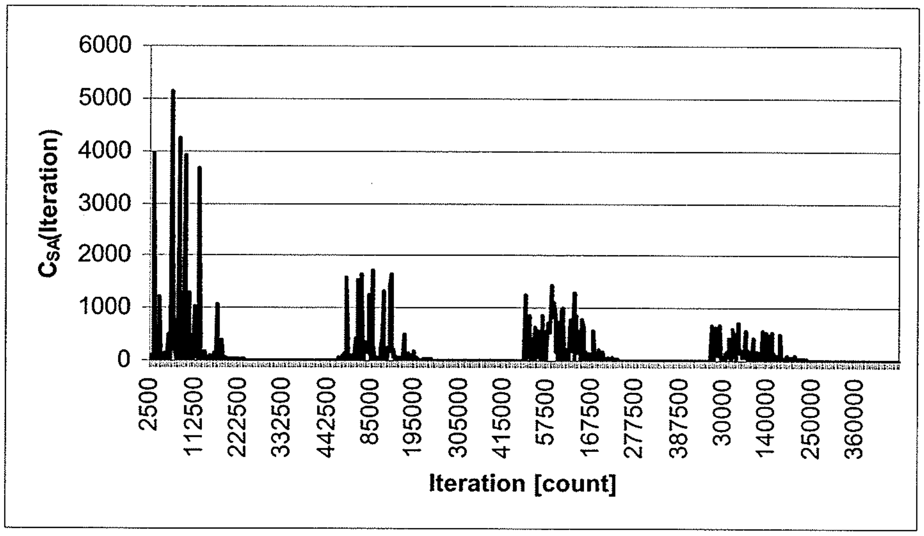


Fig. A.15 Cost plot for 1.8 cm tumour.

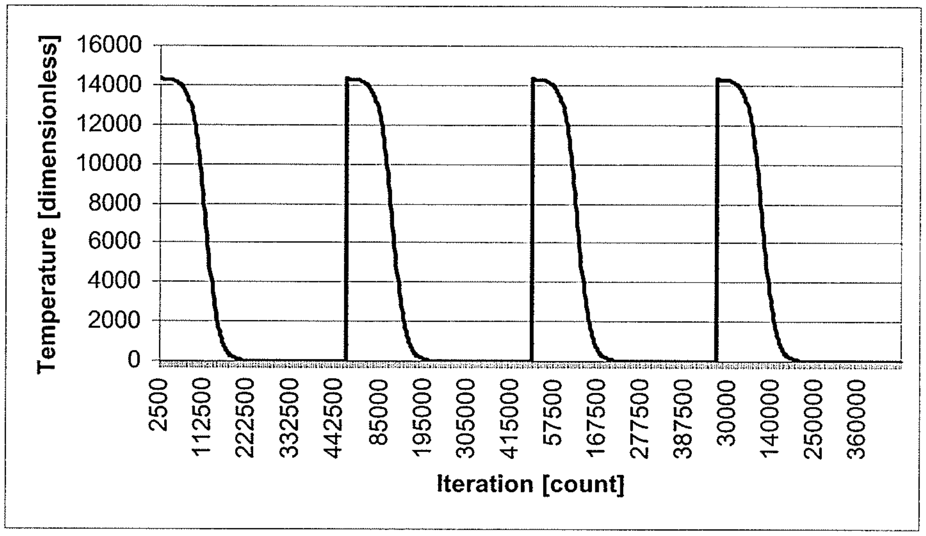


Fig. A.16 Cooling profile for 1.8 cm tumour.

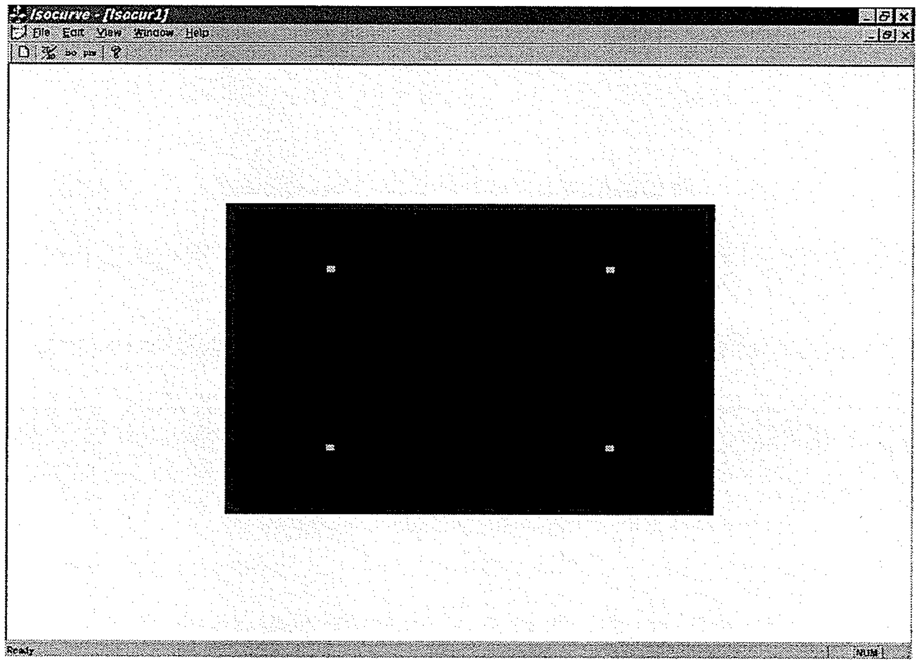


Fig. A.17 Optimized source positions for 1.8 cm tumour.

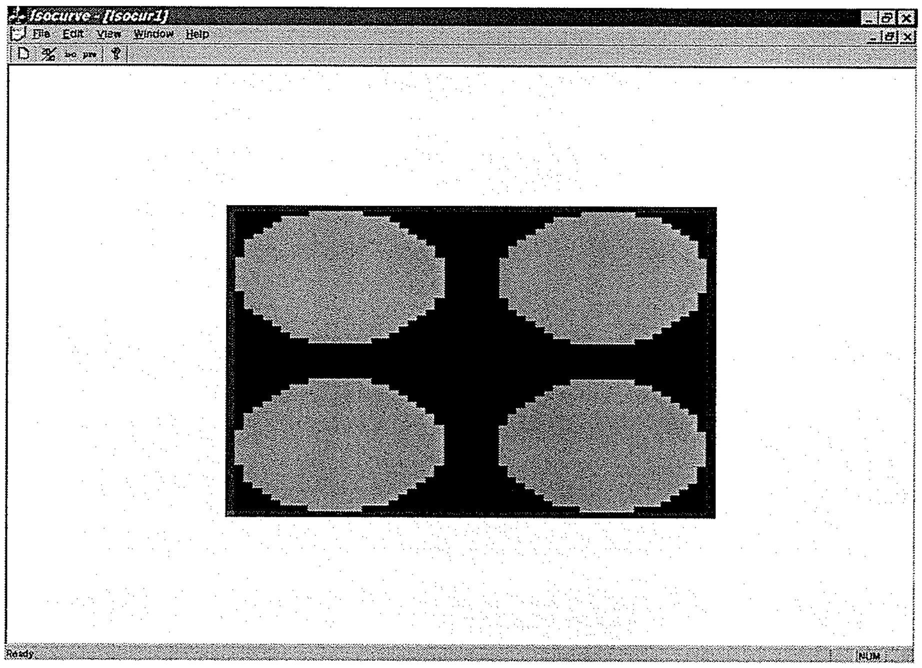


Fig. A.18 Hyperdose sleeve for 1.8 cm solution.

A.4 Square Tumour 2.2 cm in Width

The 2.2 cm tumour shown in Fig. A.19 was used as input to the SAB software. The configuration file shown in Fig. A.20 was used to control the flow of the SAB software as discussed in Section 4.3.1.2. This trial produced a cost iteration plot as shown in Fig. A.21. A cooling profile shown in Fig. A.22 was utilized. The output from the SAB software is as shown in Fig. A.23. The hyperdose sleeve for the solution shown in Fig. A.23, is shown in Fig. A.24. Although the hyperdose sleeves touch, they are actually very close to the desired size. When measured, they are found to be 1.066 cm. If we plot the 205% isodose curve, then the hyperdose sleeves no longer touch indicating we are very close to the 200% requirement. The final cost function value for this source distribution was 1.103910 however, if we allow the number of sources to increase, the SAB output for five sources has a final cost value of 1.153827 larger than that of 4 sources. The output from the SAB algorithm for a tumour size of 2.2 cm is therefore shown to be correct.

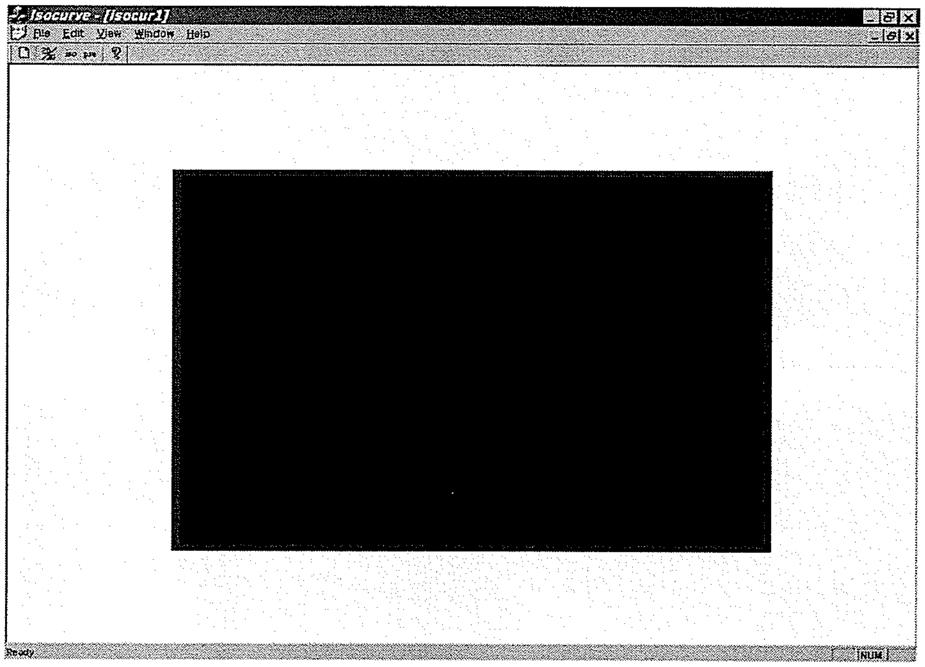
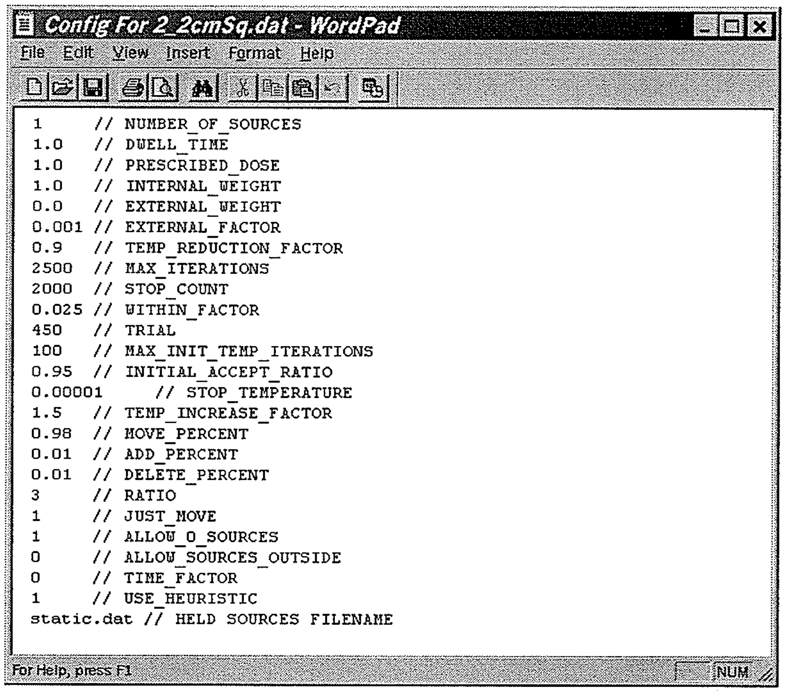


Fig. A.19 2.2 cm tumour to be optimized with SAB.


Config For 2_2cmSq.dat - WordPad
File Edit View Insert Format Help
For Help, press F1
NUM

```
1      // NUMBER_OF_SOURCES
1.0    // DWELL_TIME
1.0    // PRESCRIBED_DOSE
1.0    // INTERNAL_WEIGHT
0.0    // EXTERNAL_WEIGHT
0.001 // EXTERNAL_FACTOR
0.9    // TEMP_REDUCTION_FACTOR
2500  // MAX_ITERATIONS
2000  // STOP_COUNT
0.025 // WITHIN_FACTOR
450    // TRIAL
100    // MAX_INIT_TEMP_ITERATIONS
0.95  // INITIAL_ACCEPT_RATIO
0.00001     // STOP_TEMPERATURE
1.5    // TEMP_INCREASE_FACTOR
0.98  // MOVE_PERCENT
0.01  // ADD_PERCENT
0.01  // DELETE_PERCENT
3      // RATIO
1      // JUST_MOVE
1      // ALLOW_0_SOURCES
0      // ALLOW_SOURCES_OUTSIDE
0      // TIME_FACTOR
1      // USE_HEURISTIC
static.dat // HELD SOURCES FILENAME
```

Fig. A.20 Configuration file for 2.2 cm tumour.

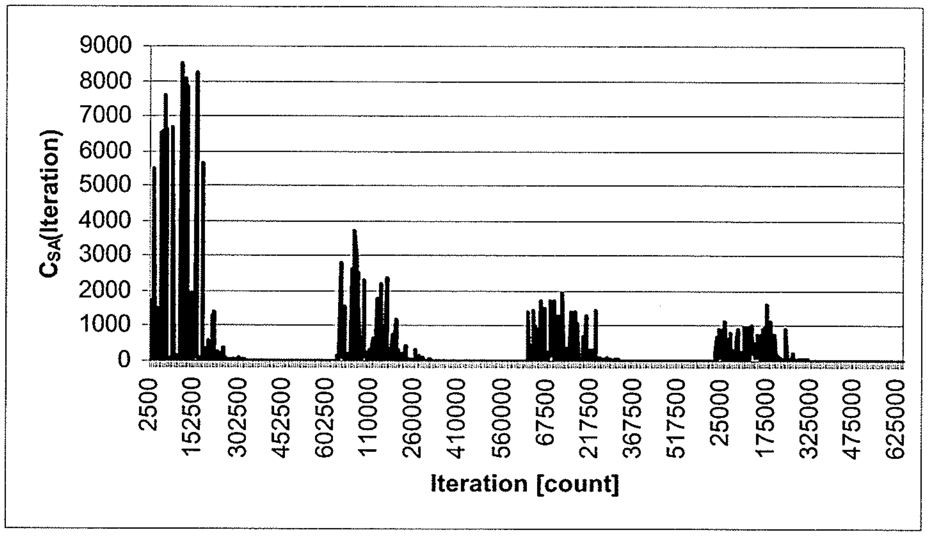


Fig. A.21 Cost plot for 2.2 cm tumour.

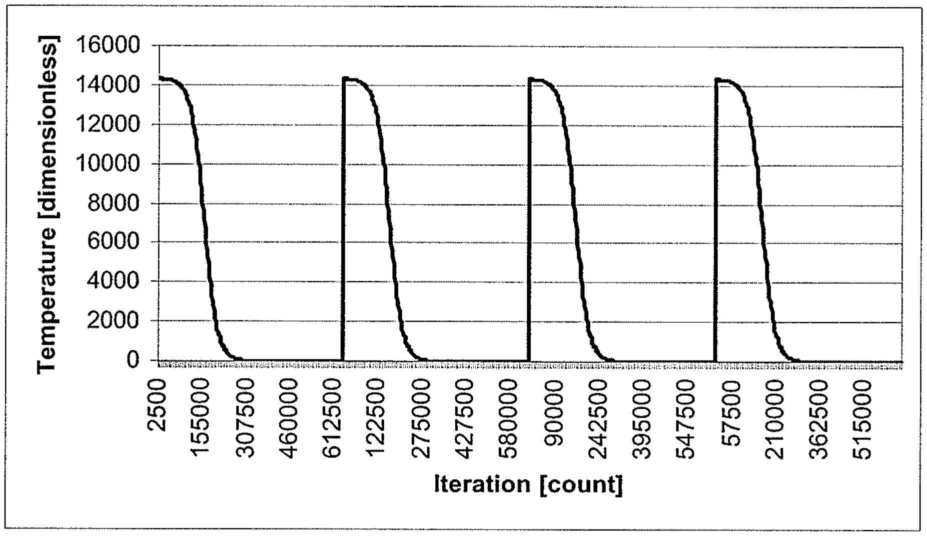


Fig. A.22 Cooling profile for 2.2 cm tumour.

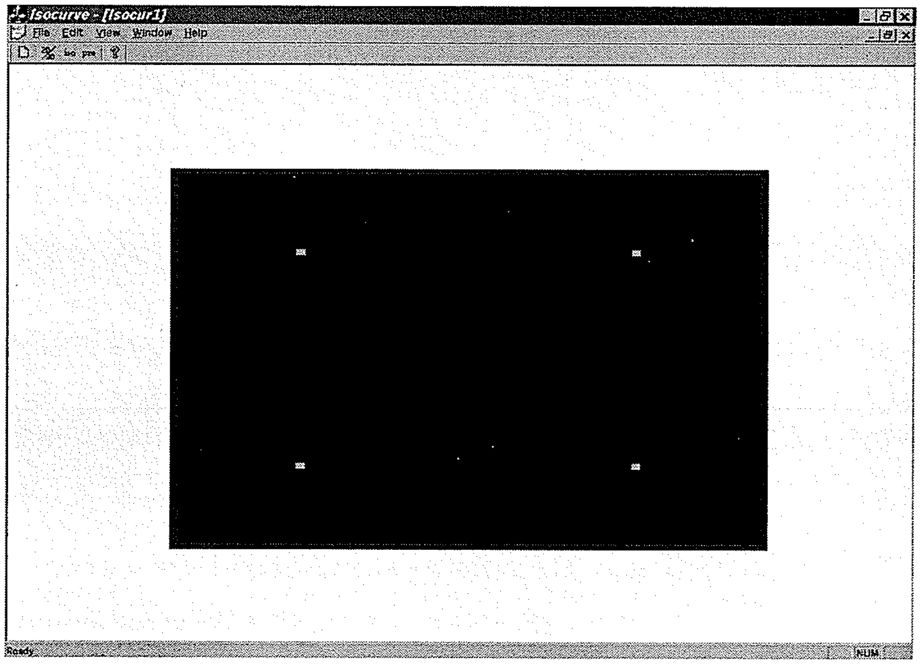


Fig. A.23 Optimized source positions for 2.2 cm tumour.

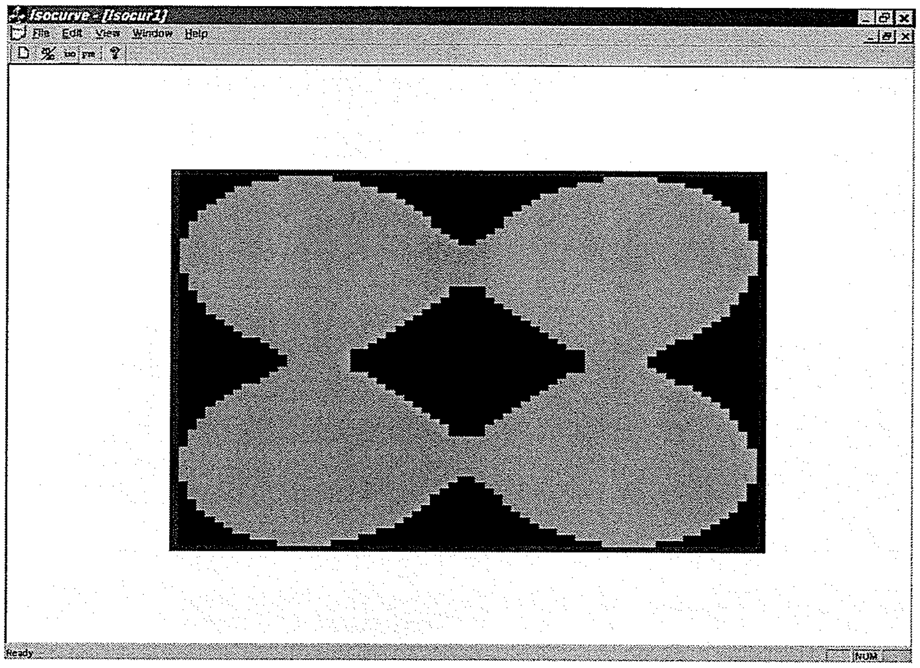


Fig. A.24 Hyperdose sleeve for 2.2 cm solution.

A.5 Square Tumour 2.8 cm in Width

The 2.8 cm tumour shown in Fig. A.25 was used as input to the SAB software. The configuration file shown in Fig. A.26 was used to control the flow of the SAB software as discussed in Section 4.3.1.2. This trial produced a cost iteration plot as shown in Fig. A.27. A cooling profile shown in Fig. A.28 was utilized. The output from the SAB software is as shown in Fig. A.29. The hyperdose sleeve for the solution shown in Fig. A.29, is shown in Fig. A.30. The results shown in Fig. A.30 have hyperdose sleeves that measure 1.066 cm. The hyperdose sleeves are larger than the desired 1.0 cm. However, when we inspect the other output generated by SAB for larger source counts, none of them have isodose sleeves as small as this solution. Therefore, SAB has generated the correct solution. The final cost function value for this source distribution was 1.034577.

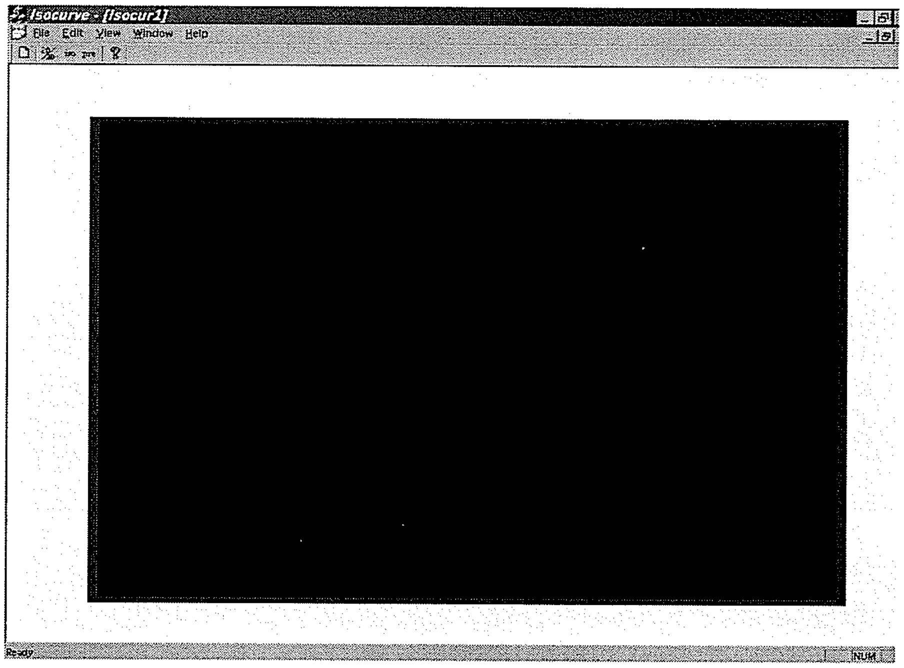
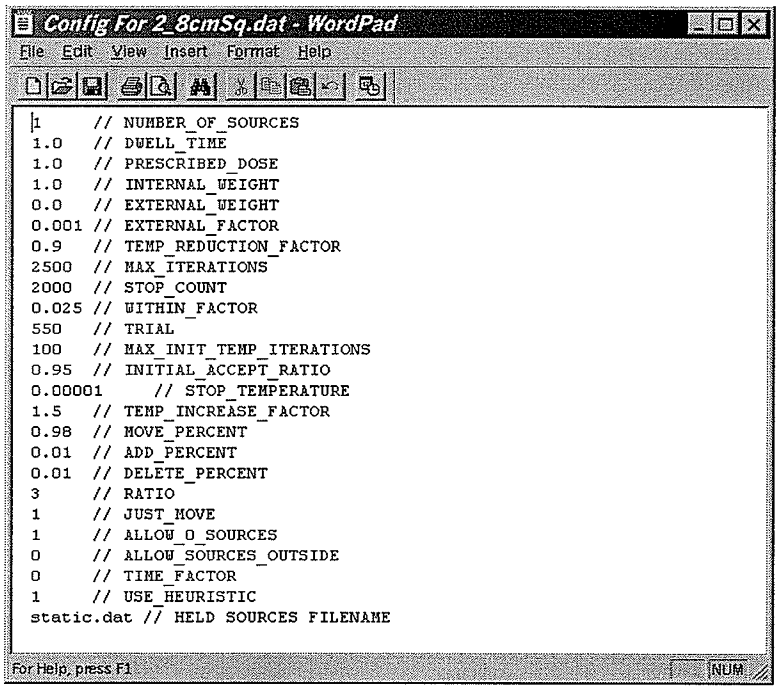


Fig. A.25 2.8 cm tumour to be optimized with SAB.


Config For 2_8cmSq.dat - WordPad
File Edit View Insert Format Help
For Help, press F1
NUM

```
1     // NUMBER_OF_SOURCES
1.0   // DWELL_TIME
1.0   // PRESCRIBED_DOSE
1.0   // INTERNAL_WEIGHT
0.0   // EXTERNAL_WEIGHT
0.001 // EXTERNAL_FACTOR
0.9   // TEMP_REDUCTION_FACTOR
2500  // MAX_ITERATIONS
2000  // STOP_COUNT
0.025 // WITHIN_FACTOR
550   // TRIAL
100   // MAX_INIT_TEMP_ITERATIONS
0.95  // INITIAL_ACCEPT_RATIO
0.00001     // STOP_TEMPERATURE
1.5   // TEMP_INCREASE_FACTOR
0.98  // MOVE_PERCENT
0.01  // ADD_PERCENT
0.01  // DELETE_PERCENT
3     // RATIO
1     // JUST_MOVE
1     // ALLOW_0_SOURCES
0     // ALLOW_SOURCES_OUTSIDE
0     // TIME_FACTOR
1     // USE_HEURISTIC
static.dat // HELD SOURCES FILENAME
```

Fig. A.26 Configuration file for 2.8 cm tumour.

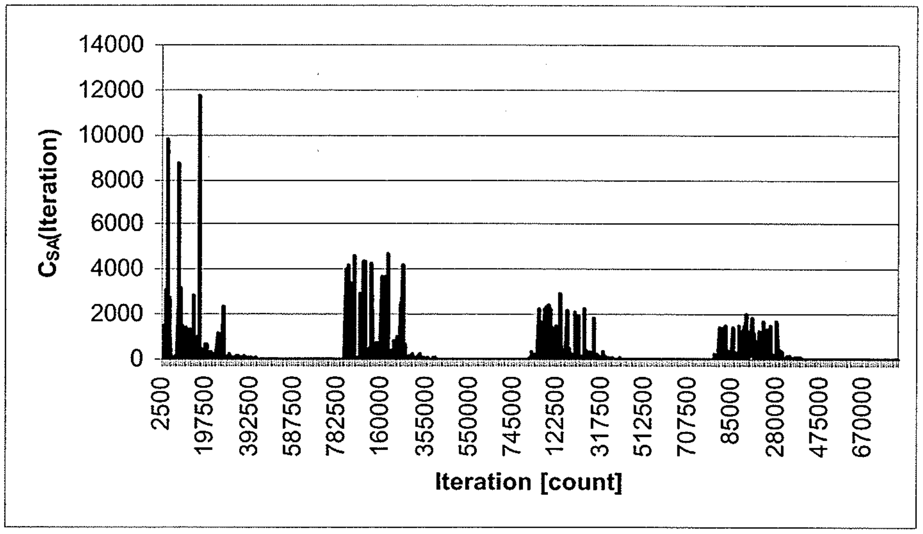


Fig. A.27 Cost plot for 2.8 cm tumour.

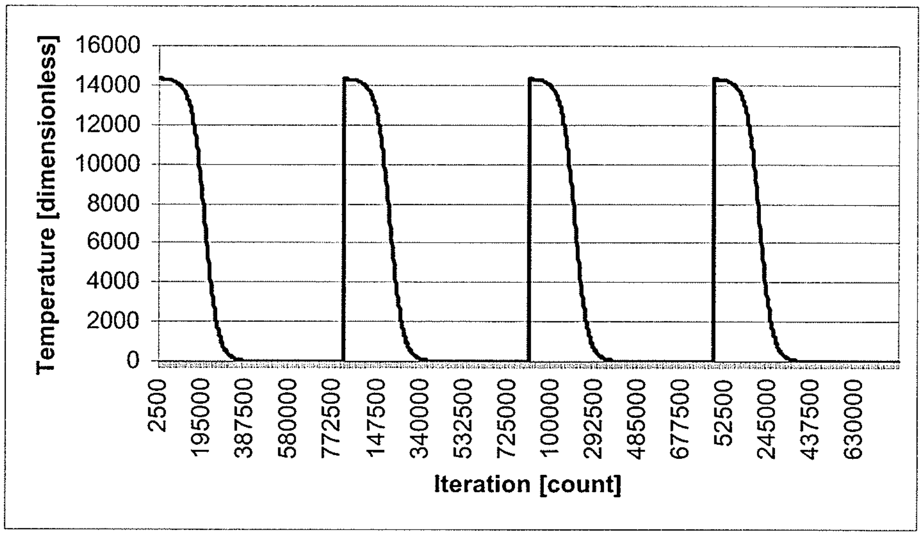


Fig. A.28 Cooling profile for 2.8 cm tumour.

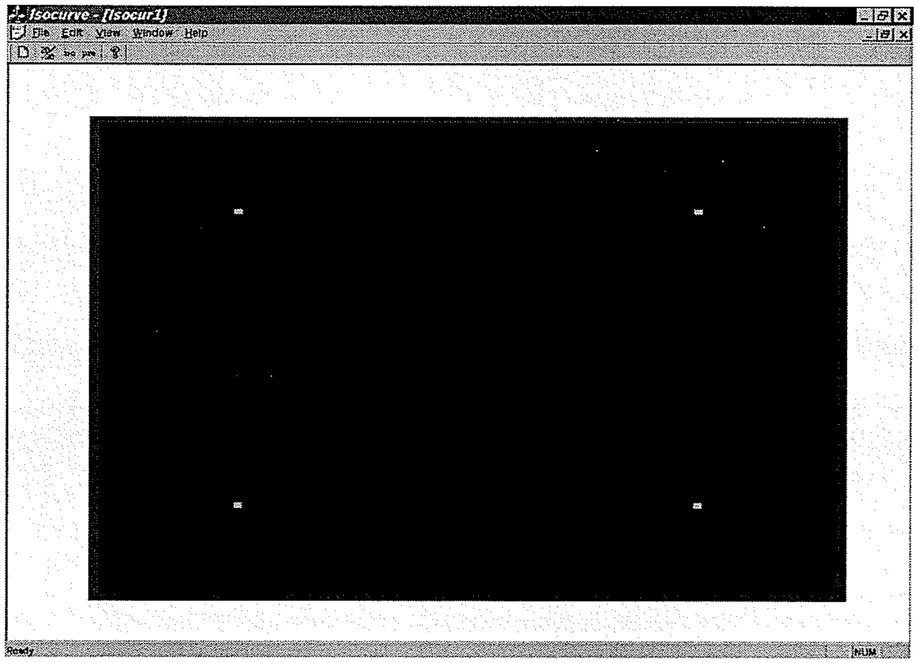


Fig. A.29 Optimized source positions for 2.8 cm tumour.

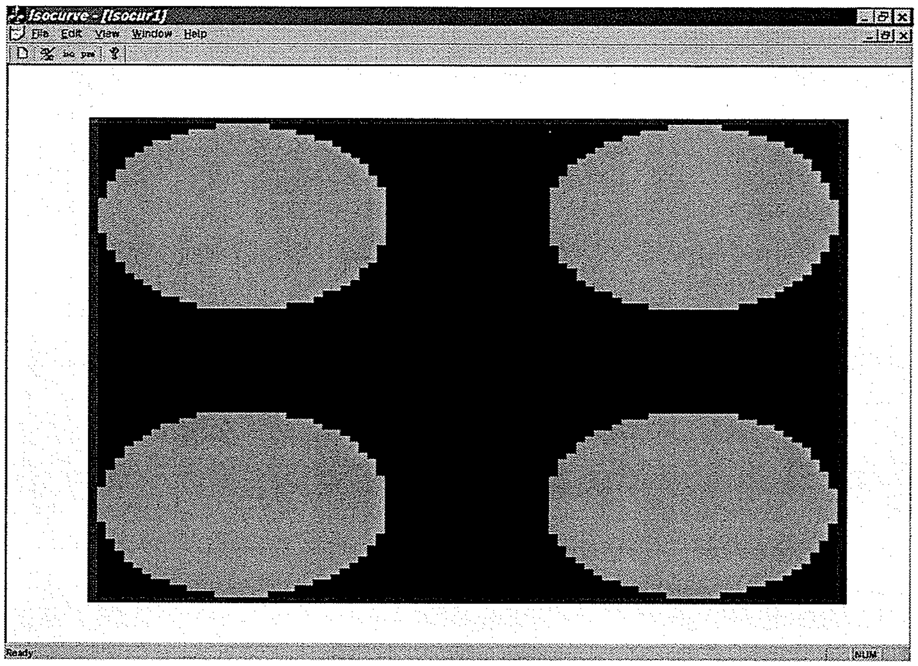


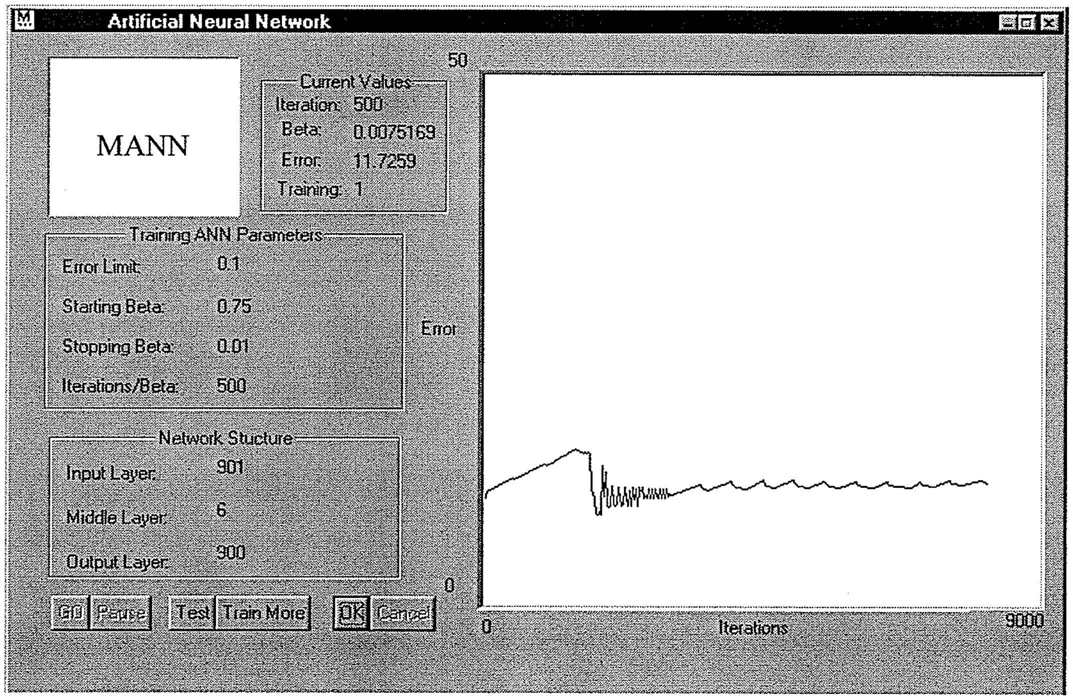
Fig. A.30 Hyperdose sleeve for 2.8 cm solution.

APPENDIX B

ADDITIONAL MANN RESULTS

B.1 ANN With 6 Internal Nodes

When the MANN software is used to train an ANN with 901 input nodes, 6 internal nodes and 900 output nodes, the results of the training are shown in Fig. B.1.


Artificial Neural Network

MANN

Current Values
Iteration: 500
Beta: 0.0075163
Error: 11.7259
Training: 1

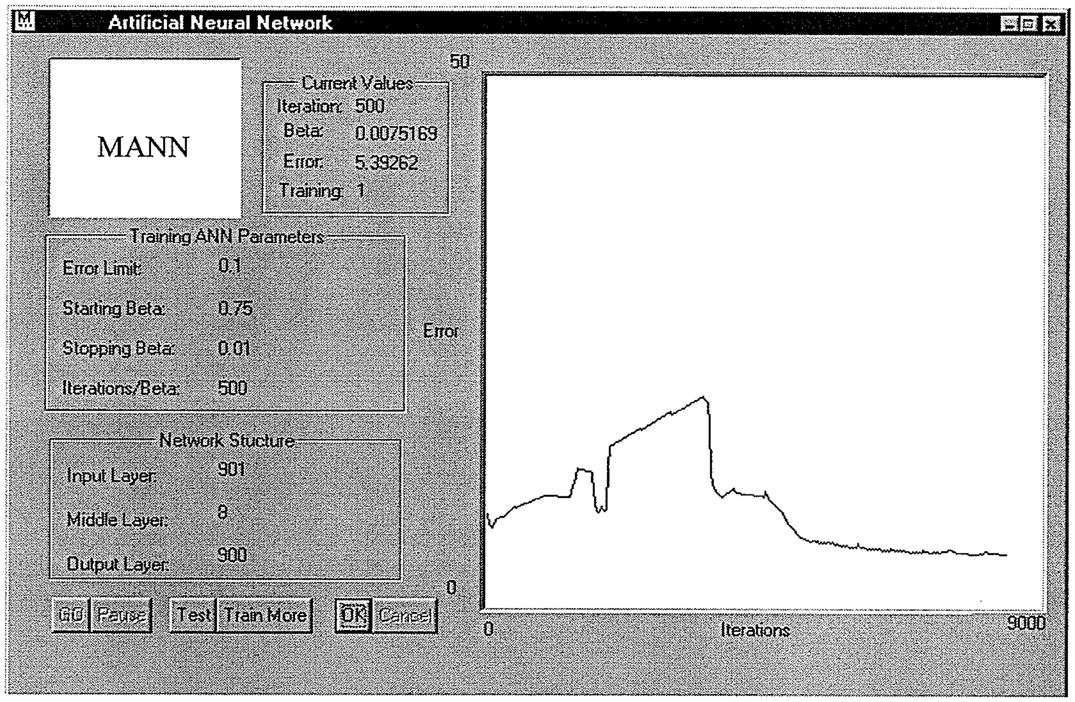
Training ANN Parameters
Error Limit: 0.1
Starting Beta: 0.75
Stopping Beta: 0.01
Iterations/Beta: 500

Network Stucture
Input Layer: 901
Middle Layer: 6
Output Layer: 900

Fig. B.1 Results of training ANN using MANN for 6 internal nodes.

B.2 ANN With 8 Internal Nodes

When the MANN software is used to train an ANN with 901 input nodes, 8 internal nodes and 900 output nodes, the results of the training are shown in Fig. B.1.



Current Values
Iteration: 500
Beta: 0.0075169
Error: 5.39262
Training: 1

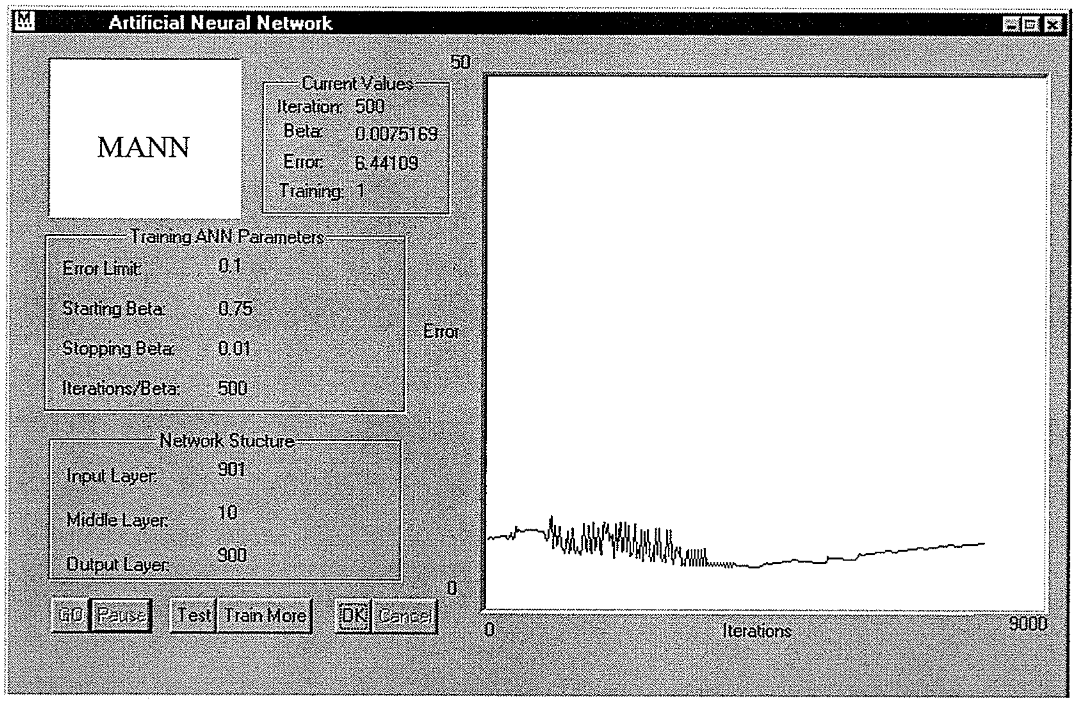
Training ANN Parameters
Error Limit: 0.1
Starting Beta: 0.75
Stopping Beta: 0.01
Iterations/Beta: 500

Network Stucture
Input Layer: 901
Middle Layer: 8
Output Layer: 900

Fig. B.2 Results of training ANN using MANN for 8 internal nodes.

B.3 ANN With 10 Internal Nodes

When the MANN software is used to train an ANN with 901 input nodes, 10 internal nodes and 900 output nodes, the results of the training are shown in Fig. B.2.



Current Values
Iteration: 500
Beta: 0.0075169
Error: 6.44109
Training: 1

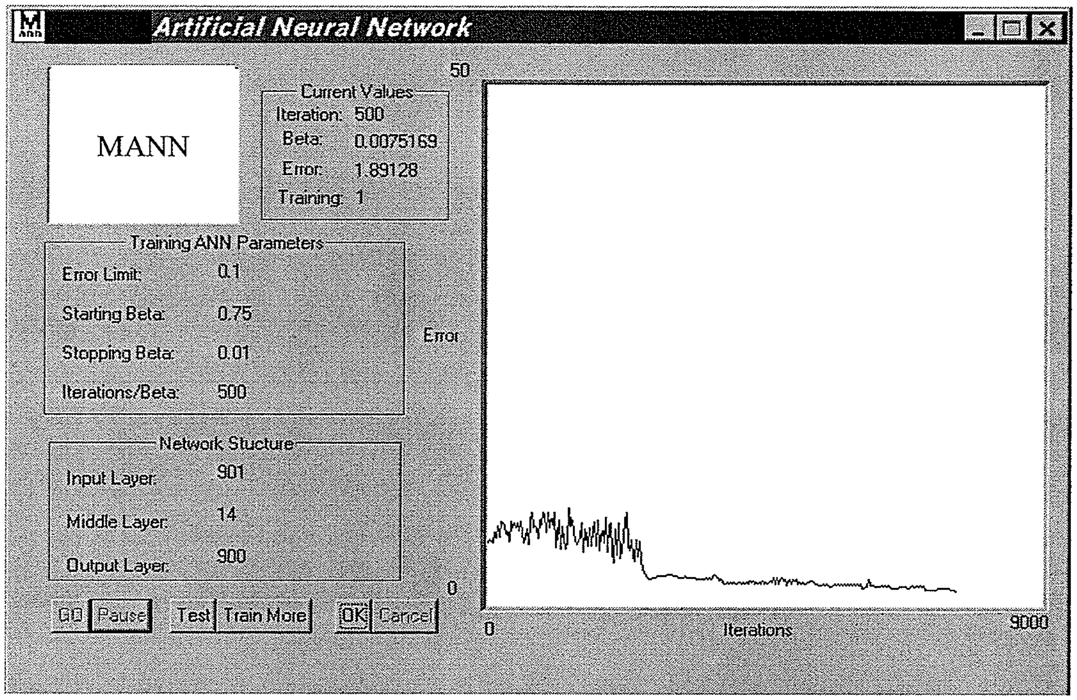
Training ANN Parameters
Error Limit: 0.1
Starting Beta: 0.75
Stopping Beta: 0.01
Iterations/Beta: 500

Network Stucture
Input Layer: 901
Middle Layer: 10
Output Layer: 900

Fig. B.3 Results of training ANN using MANN for 10 internal nodes.

B.4 ANN With 14 Internal Nodes

When the MANN software is used to train an ANN with 901 input nodes, 14 internal nodes and 900 output nodes, the results of the training are shown in Fig. B.4.



Current Values
Iteration: 500
Beta: 0.0075169
Error: 1.89128
Training: 1

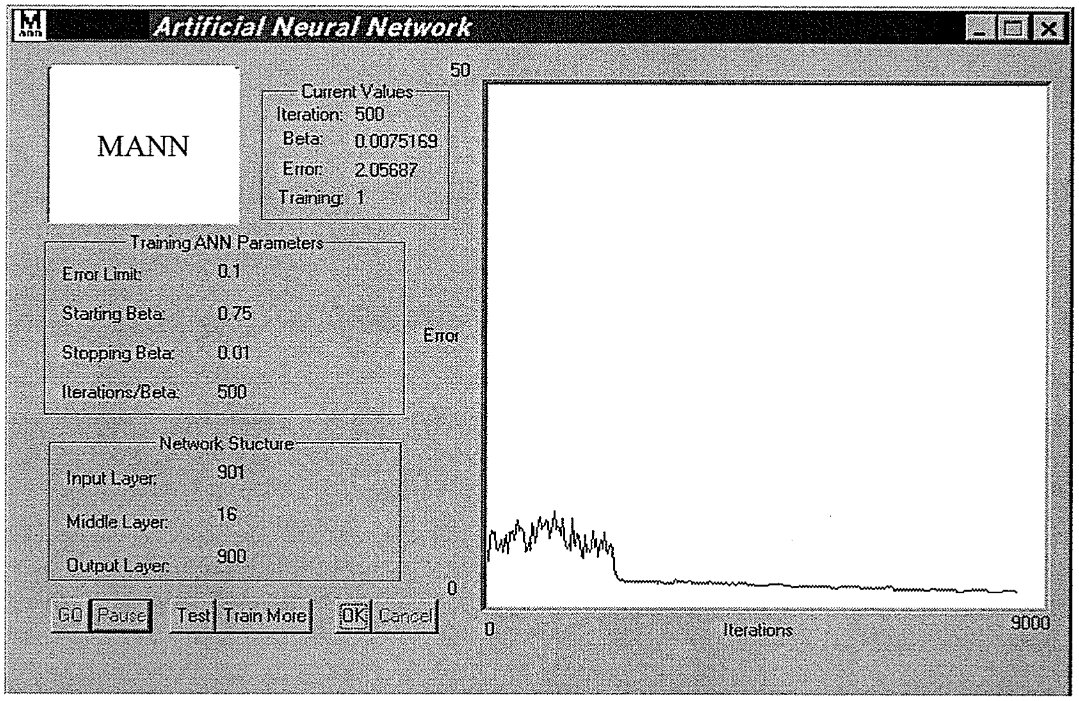
Training ANN Parameters
Error Limit: 0.1
Starting Beta: 0.75
Stopping Beta: 0.01
Iterations/Beta: 500

Network Stucture
Input Layer: 901
Middle Layer: 14
Output Layer: 900

Fig. B.4 Results of training ANN using MANN for 14 internal nodes.

B.5 ANN With 16 Internal Nodes

When the MANN software is used to train an ANN with 901 input nodes, 16 internal nodes and 900 output nodes, the results of the training are shown in Fig. B.5.



Current Values
Iteration: 500
Beta: 0.0075169
Error: 2.05687
Training: 1

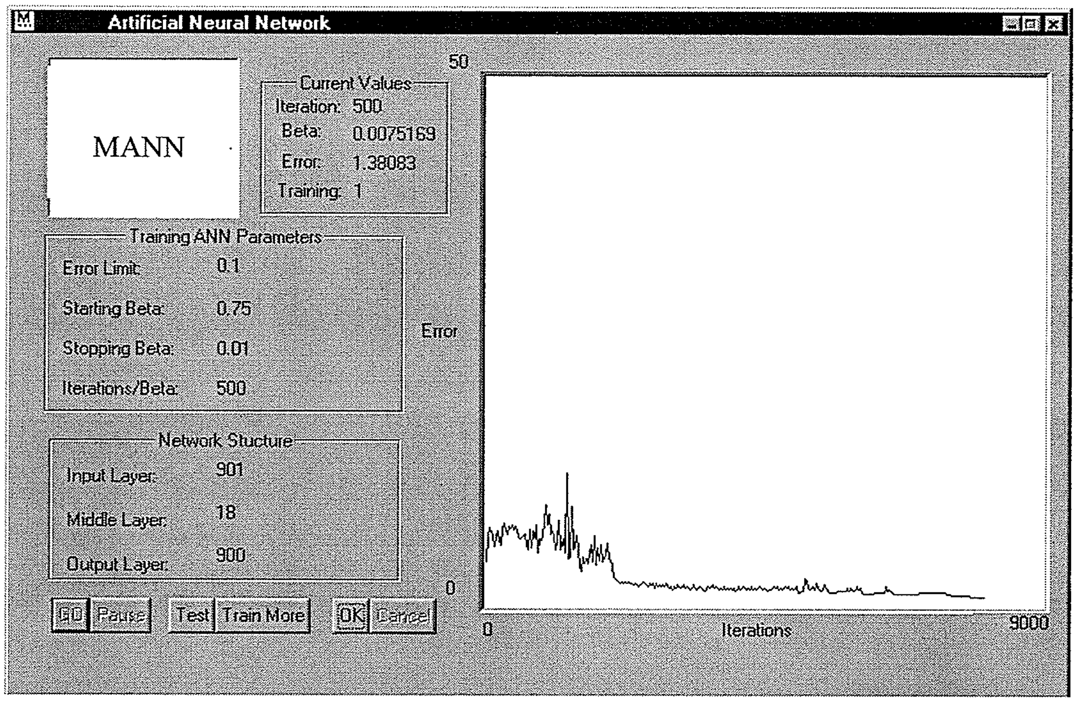
Training ANN Parameters
Error Limit: 0.1
Starting Beta: 0.75
Stopping Beta: 0.01
Iterations/Beta: 500

Network Stucture
Input Layer: 901
Middle Layer: 16
Output Layer: 900

Fig. B.5 Results of training ANN using MANN for 16 internal nodes.

B.6 ANN With 18 Internal Nodes

When the MANN software is used to train an ANN with 901 input nodes, 18 internal nodes and 900 output nodes, the results of the training are shown in Fig. B.6.


Artificial Neural Network

MANN

Current Values
Iteration: 500
Beta: 0.0075169
Error: 1.38083
Training: 1

Training ANN Parameters
Error Limit: 0.1
Starting Beta: 0.75
Stopping Beta: 0.01
Iterations/Beta: 500

Network Stucture
Input Layer: 901
Middle Layer: 18
Output Layer: 900

Fig. B.6 Results of training ANN using MANN for 18 internal nodes.

APPENDIX C
SAB SOURCE FILES

C.1 Cost.c

```
/***************************
FILENAME:          cost.c
PROJECT:                SAB - Simulated Annealing for Brachytherapy
AUTHOR:                 S. Miller
DESCRIPTION:    Implements functions to calculate the cost
                        of a specific treatment in a tumour
ASSUMPTIONS:    none
DATE WRITTEN:   Summer 1998 - Summer 2002
MODIFICATION HISTORY:Version 1.0
*/
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "globals.h"


/*******************************************
FUNCTION NAME:CheckAreaRecursive
PURPOSE:        To recursively go through a tumour and treatment map
                        and figure out the area of the hyperdose (dose>200%)
INPUT:          The address of the treatment map, the x, and y coordinates
                        and the address of the count of the area (in pixels)
OUTPUT:         None
FUNCTIONS CALLED:CheckAreaRecursive
ASSUMPTIONS:The treatment map has already been converted into a binary representation,
                        where 1 means dose is greater than 200%, and 0 means dose is less than
                        200%.

        NOTE:  This is a recursive function - watch stack usage!
*/
void CheckAreaRecursive(DoseArea * Array,int x,int y,int * count)
{
    if ((Array +x*xSize + y)->Dose > 0.0f)
    {
        //set it to 0.0f because we have included it now
        (Array + x*xSize + y)->Dose = 0.0f;

        //increase the count of the area
        (*count)++;

        //find the rest of the 1.0s in the area
        if (x>0)
            CheckAreaRecursive(Array,x-1,y,count);
        if (y>0)
            CheckAreaRecursive(Array,x,y-1,count);
        if (x<xSize-1)
            CheckAreaRecursive(Array,x+1,y,count);
        if (y<ySize-1)
            CheckAreaRecursive(Array,x,y+1,count);
    }//end if dose>0
}//end function CheckAreaRecursive


/*******************************************
FUNCTION NAME:CheckHyperDoseSleaveIntegration
PURPOSE:        Checks if all of the hyperdose sleaves are within spec - i.e.
                        are only 1cm squared
```

```
INPUT:          The address of the current treatment solution - point source list
OUTPUT:         1 if hyperdose sleaves are OK, else 0.
FUNCTIONS CALLED:CheckAreaRecursive
ASSUMPTIONS:The treatment map has already been converted into a binary representation,
                        where 1 means dose is greater than 200%, and 0 means dose is less than
                        200%.
*/
int CheckHyperDoseSleaveIntegration(PS *ThePS)
{
    int i=0,j=0;// for loop variables
    float Dose=0.f;// dose at each point in the tumour map
    PS *TempPS = NULL;// temporary point source solution list
    PS *TempStaticPS=NULL;// temp to hold static point sources
    float Min=9999999.f;// the minimum dose in the tumour
    DoseArea *TempArea=NULL;/* This is the 2D array of the tumour area */
    int count=0;// count of area size
    float mmSize=0.f;// holds the milli meter area size
    float maxSize=0.f;//holds the maximum allowed area size

    /*Go through the whole "Area" and calculate the dose*/
    for (i=0;i<xSize;i++)
    {
        for (j=0;j<ySize;j++)
        {
            // Run though Dynamic list of Sources
                Dose = 0.0f;
                TempPS = ThePS;
                while (TempPS)
                {
                    if ((TempPS -> x == i) && (TempPS -> y == j))
                        Dose += TempPS -> DwellTime*sqr(RATIO);
                    else
                        Dose += ((TempPS -> DwellTime)*sqr(RATIO))/sqr(distance(i,j,TempPS->x,TempPS-
>y));

                    TempPS = TempPS -> NextPS;
                }// end while

                // Run through Static list of Sources
                TempStaticPS = FirstStaticPS;
                while (TempStaticPS)
                {
                    if ((TempStaticPS -> x == i) && (TempStaticPS -> y == j))
                        Dose += TempStaticPS -> DwellTime * sqr(RATIO);
                    else
                        Dose += ((TempStaticPS -> DwellTime)*sqr(RATIO))/sqr(distance(i,j,TempStaticPS-
>x,TempStaticPS->y));

                    TempStaticPS = TempStaticPS -> NextPS;
                }// end while

                // Store the Dose to the "Area" array
                (Area + ySize*i + j) -> Dose = Dose;

                // might as well store the minimum to the periphery at the same time
                if (((Area + ySize*i + j) -> Type == PERIPHERY)&&(Dose<Min))
                    Min = Dose;
        }//end for j
    }//end for i

    // 5 steps to check hyperdoses
    //1. Make new area as exact copy of old area and set any point receiving
    // too much dose (200% or more) to 1, and set the rest to 0
    TempArea = (DoseArea *)malloc(xSize*ySize*sizeof(DoseArea));
    for (i=0;i<xSize;i++)
    {
        for (j=0;j<ySize;j++)
        {
            if (((Area+i*ySize+j)->Dose/(float)Min) > 2.0f)
                (TempArea+i*ySize+j)->Dose = 1.0f;
            else
                (TempArea+i*ySize+j)->Dose = 0.0f;
            (TempArea+i*ySize+j)->Type = (Area+i*ySize+j)->Type;
        }
    }

    //2. Use recursive algorithm to find sizes of hyperdose sleaves using map
    // created in step 1
    fprintf(stderr,"There are %d sources\n",NumPS(ThePS));
    for (i=0;i<xSize;i++)
    {
        for (j=0;j<ySize;j++)
        {
            count=0;
            CheckAreaRecursive(TempArea,i,j,&count);
            //3. turn count into mm*mm
            mmSize = (float)count/(float)(RATIO*RATIO);
            maxSize = (float)78.53981634; //5*5*PI (PI * r sqrd, in mm)

            if (count>0)
                fprintf(stderr,"(%d,%d) mmSize is: %f max size is %f\n",i,j,mmSize,maxSize);

            //4. return(1) if bad
            if (mmSize>maxSize)
            {
                fprintf(stderr,"Did not reach the end of checkhyperdosesleaveintegration, hyperdose
sleave too big.\n\n");
                return(1);//the hyperdose area is too big
            }//end if hyperdose size is too big
        }//end for j
    }//end for i

    //5. Else return(0) good
    fprintf(stderr,"All GoOoD in CheckHyperDoseSleaveIntegration\n\n");
    return(0);
}//end function CheckHyperDoseSleaveIntegration



/*******************************************
FUNCTION NAME:CheckHyperDoseSleave
PURPOSE:        Checks if all of the hyperdose sleaves are within spec - i.e.
                        are only 1cm squared
INPUT:          The address of the current treatment solution - point source list
OUTPUT:         1 if hyperdose sleaves are NOT OK, else 0 if OK
FUNCTIONS CALLED:CheckHyperDoseSleaveIntegration
ASSUMPTIONS:None
*/
int CheckHyperDoseSleave(PS *ThePS)
{
    int i;  // for loop index
    int j;  // for loop index
    float Dose=0.0f;// holds the dose at a point
    PS *TempPS=NULL;// temp list of point sources
    PS *TempStaticPS=NULL;// temp list of static point sources
    float Min=9999999.f;// minimum dose in tumour so far

    /*Go through the whole "Area" and calculate the dose*/
    for (i=0;i<xSize;i++)
    {
        for (j=0;j<ySize;j++)
        {
            // Run though Dynamic list of Sources
            Dose = 0.0f;
            TempPS = ThePS;
            while (TempPS)
            {
                if ((TempPS -> x == i) && (TempPS -> y == j))
                    Dose += TempPS -> DwellTime*sqr(RATIO);
                else
                    Dose += ((TempPS -> DwellTime)*sqr(RATIO))/sqr(distance(i,j,TempPS->x,TempPS->y));
                TempPS = TempPS -> NextPS;
            }// end while

            // Run through Static list of Sources
            TempStaticPS = FirstStaticPS;
            while (TempStaticPS)
            {
                if ((TempStaticPS -> x == i) && (TempStaticPS -> y == j))
                    Dose += TempStaticPS -> DwellTime * sqr(RATIO);
                else
                    Dose += ((TempStaticPS -> DwellTime)*sqr(RATIO))/sqr(distance(i,j,TempStaticPS-
>x,TempStaticPS->y));

                TempStaticPS = TempStaticPS -> NextPS;
            }// end while

            // Store the Dose to the "Area" array
            (Area + ySize*i + j) -> Dose = Dose;

            // might as well store the minimum to the periphery at the same time
            if (((Area + ySize*i + j) -> Type == PERIPHERY)&&(Dose<Min))
                Min = Dose;
        }//end for j
    }//end for i


    //check that the hyperdose sleave is small enough for Dynamic Sources
    TempPS = ThePS;
    while (TempPS)
    {
        j = TempPS -> y;
        i = TempPS -> x;
        /* west */
        if (j>5*RATIO)
        {
            Dose = (((Area + ySize*i + (j-5*RATIO))->Dose)/Min);
            if (Dose>2.f)
                return(1);
        }
        else //j=0
        {
            Dose = (((Area + ySize*i)->Dose)/Min);
            if (Dose>2.f)
                return(1);
        }
        /* north */
        if (i>5*RATIO)
        {
            Dose = (((Area + ySize*(i-5*RATIO) + j)->Dose)/Min);
            if (Dose>2.f)
                return(1);
        }
        else //i=0
        {
            Dose = (((Area + j)->Dose)/Min);
            if (Dose>2.f)
                return(1);
        }
        /* east */
        if (j+5*RATIO<ySize)
        {
            Dose = (((Area + ySize*i + (j+5*RATIO))->Dose)/Min);
            if (Dose>2.f)
                return(1);
        }
        else //j=ySize-1
        {
            Dose = (((Area + ySize*i + (ySize-1))->Dose)/Min);
            if (Dose>2.f)
                return(1);
        }
        /* south */
        if (i+5*RATIO<xSize)
        {
            Dose = (((Area + ySize*(i+5*RATIO) + j)->Dose)/Min);
            if (Dose>2.f)
            return(1);
        }
        else // i = xSize-1
        {
            Dose = (((Area + ySize*(xSize-1) + j)->Dose)/Min);
            if (Dose>2.f)
                return(1);
        }
        TempPS = TempPS -> NextPS;
    }/* end while */


    //check that the hyperdose sleave is small enough for Static Sources
    TempPS = FirstStaticPS;
    while (TempPS)
    {
        j = TempPS -> y;
        i = TempPS -> x;
        /* west */
        if (j>5*RATIO)
        {
            Dose = (((Area + ySize*i + (j-5*RATIO))->Dose)/Min);
            if (Dose>2.f)
                return(1);
        }
        else //j=0
        {
            Dose = (((Area + ySize*i)->Dose)/Min);
            if (Dose>2.f)
                return(1);
        }
        /* north */
        if (i>5*RATIO)
        {
            Dose = (((Area + ySize*(i-5*RATIO) + j)->Dose)/Min);
            if (Dose>2.f)
                return(1);
        }
        else //i=0
        {
            Dose = (((Area + j)->Dose)/Min);
            if (Dose>2.f)
                return(1);
        }
        /* east */
        if (j+5*RATIO<ySize)
        {
            Dose = (((Area + ySize*i + (j+5*RATIO))->Dose)/Min);
            if (Dose>2.f)
                return(1);
        }
        else //j=ySize-1
        {
            Dose = (((Area + ySize*i + (ySize-1))->Dose)/Min);
            if (Dose>2.f)
                return(1);
        }
        /* south */
        if (i+5*RATIO<xSize)
        {
            Dose = (((Area + ySize*(i+5*RATIO) + j)->Dose)/Min);
            if (Dose>2.f)
                return(1);
        }
        else // i = xSize-1
        {
            Dose = (((Area + ySize*(xSize-1) + j)->Dose)/Min);
            if (Dose>2.f)
                return(1);
        }
        TempPS = TempPS -> NextPS;
    }/* end while */


/*
//check that the hyperdose sleave is BIG enough
    TempPS = ThePS;
    while (TempPS)
    {
        j = TempPS -> y;
        i = TempPS -> x;
        // west
        if (j>4)
        {
            Dose = (((Area + ySize*i + (j-4))->Dose)/Min);
            if (Dose<=2.f)
                return(-1);
        }
        else //j=0
        {
            Dose = (((Area + ySize*i)->Dose)/Min);
            if (Dose<=2.f)
                return(-1);
        }
        // north
        if (i>4)
        {
            Dose = (((Area + ySize*(i-4) + j)->Dose)/Min);
            if (Dose<=2.f)
                return(-1);
        }
        else //i=0
        {
            Dose = (((Area + j)->Dose)/Min);
            if (Dose<=2.f)
                return(-1);
        }
        // east
        if (j+4<ySize)
        {
            Dose = (((Area + ySize*i + (j+4))->Dose)/Min);
            if (Dose<=2.f)
                return(-1);
        }
        else //j=ySize-1
        {
            Dose = (((Area + ySize*i + (ySize-1))->Dose)/Min);
            if (Dose<=2.f)
                return(-1);
        }
        // south
        if (i+4<xSize)
        {
            Dose = (((Area + ySize*(i+4) + j)->Dose)/Min);
            if (Dose<=2.f)
                return(-1);
        }
        else // i = xSize-1
        {
            Dose = (((Area + ySize*(xSize-1) + j)->Dose)/Min);
            if (Dose<=2.f)
                return(-1);
        }
        TempPS = TempPS -> NextPS;
    }// end while
*/

    // if nothing else has a problem, return that all is well (0)
    return(0);
}//end function CheckHyperDoseSleave
```

```
/*******************************************
FUNCTION NAME:CalculateCost
PURPOSE:        Calcualtes the cost of the current solution - this is the COST FUNCTION
                        for SAB!
INPUT:          The address of the current treatment solution - point source list
OUTPUT:         The cost of the current solution
FUNCTIONS CALLED:None
ASSUMPTIONS:None
*/
float CalculateCost(PS *ThePS)
{
    int i;          // for loop index
    int j;          // for loop index
    float Dose=0.0f;// holds dose of current point in tumour
    PS *TempPS=NULL;// temp list of Point Sources
    PS *TempStaticPS=NULL;// temp list of static point sources
    float MinTumour=9999999.f;// minimum dose in tumour
    float MinPer = 9999999.f;// minimum peripheral dose
    float MaxPer = 0.f;// maximum peripheral dose
    float TotalDoseToHyperCheckPoints=0;// holds dose to all hyperdose checkpoints
    int Count=0;// counts number of check points > 200% to calculate average
    float A=0.f,B=0.f,C=0.f;// 3 elements of cost function
    float TempFloat=0.0f;// temp to hold output from function (cost)
    float SumDif=0.f;// holds the actual dose at the check points

    /*Go through the whole "Area" and calculate the dose*/
    for (i=0;i<xSize;i++)
    {
        for (j=0;j<ySize;j++)
        {
            // dose is initially 0
            Dose = 0.0f;

            // Run through Dynamic list of Sources
            TempPS = ThePS;
            while (TempPS)
            {
                if ((TempPS -> x == i) && (TempPS -> y == j))
                    Dose += TempPS -> DwellTime * sqr(RATIO);
                else
                    Dose += ((TempPS -> DwellTime)*sqr(RATIO))/sqr(distance(i,j,TempPS->x,TempPS->y));

                TempPS = TempPS -> NextPS;
            }//end while for dynamic sources

            // Run through Static list of Sources
            TempStaticPS = FirstStaticPS;
            while (TempStaticPS)
            {
                if ((TempStaticPS -> x == i) && (TempStaticPS -> y == j))
                    Dose += TempStaticPS -> DwellTime * sqr(RATIO);
                else
                    Dose += ((TempStaticPS -> DwellTime)*sqr(RATIO))/sqr(distance(i,j,TempStaticPS-
>x,TempStaticPS->y));

                TempStaticPS = TempStaticPS -> NextPS;
            }// end while dynamic sources

            // Store the Dose to the "Area" array
            (Area + ySize*i + j) -> Dose = Dose;

            /*might as well store the minimum to the tumour at the same time...*/
            if (((Area + ySize*i + j) -> Type == TUMOUR)&&(Dose<MinTumour))
                MinTumour = Dose;
            else if (((Area + ySize*i + j) -> Type == PERIPHERY)&&(Dose<MinPer))
                MinPer = Dose;
            else if (((Area + ySize*i + j) -> Type == PERIPHERY)&&(Dose>MaxPer))
                MaxPer = Dose;
        }//end for j
    }//end for i
```

```
/* We have the minimum to the tumour and all of the doses calculated */
/* Now, we must find the hyper dose sleave in 4 dir'ns for ===DYNAMIC===*/
TempPS = ThePS;
while (TempPS)
{
    j = TempPS -> y;
    i = TempPS -> x;
    /* west */
    if (j>5*RATIO)
    {
        Dose = (((Area + ySize*i + (j-5*RATIO))->Dose)/MinPer);
        TotalDoseToHyperCheckPoints+=Dose;
        SumDif += fabs(Dose-2.0);
        Count++;
    }
    else //j=0
    {
        Dose = (((Area + ySize*i)->Dose)/MinPer);
        TotalDoseToHyperCheckPoints+=Dose;
        SumDif += fabs(Dose-2.0);
        Count++;
    }
    /* north */
    if (i>5*RATIO)
    {
        Dose = (((Area + ySize*(i-5*RATIO) + j)->Dose)/MinPer);
        TotalDoseToHyperCheckPoints+=Dose;
        SumDif += fabs(Dose-2.0);
        Count++;}
    else //i=0
    {
        Dose = (((Area + j)->Dose)/MinPer);
        SumDif += fabs(Dose-2.0);
        TotalDoseToHyperCheckPoints+=Dose;
        Count++;
    }
    /* east */
    if (j+5*RATIO<ySize)
    {
        Dose = (((Area + ySize*i + (j+5*RATIO))->Dose)/MinPer);
        TotalDoseToHyperCheckPoints+=Dose;
        SumDif += fabs(Dose-2.0);
        Count++;
    }
    else //j=ySize-1
    {
        Dose = (((Area + ySize*i + (ySize-1))->Dose)/MinPer);
        TotalDoseToHyperCheckPoints+=Dose;
        SumDif += fabs(Dose-2.0);
        Count++;}
    /* south */
    if (i+5*RATIO<xSize)
    {
        Dose = (((Area + ySize*(i+5*RATIO) + j)->Dose)/MinPer);
        TotalDoseToHyperCheckPoints+=Dose;
        SumDif += fabs(Dose-2.0);
        Count++;}
    else // i = xSize-1
    {
        Dose = (((Area + ySize*(xSize-1) + j)->Dose)/MinPer);
        TotalDoseToHyperCheckPoints+=Dose;
        SumDif += fabs(Dose-2.0);
        Count++;
    }
    TempPS = TempPS -> NextPS;
}// end while dynamic sources


/* We have the minimum to the tumour and all of the doses calculated */
/* Now, we must find the hyper dose sleave in 4 dir'ns for ===STATIC===*/
TempPS = FirstStaticPS;
```

```
while (TempPS)
{
    j = TempPS -> y;
    i = TempPS -> x;
    /* west */
    if (j>5*RATIO)
    {
        Dose = (((Area + ySize*i + (j-5*RATIO))->Dose)/MinPer);
        TotalDoseToHyperCheckPoints+=Dose;
        SumDif += fabs(Dose-2.0);
        Count++;
    }
    else //j=0
    {
        Dose = (((Area + ySize*i)->Dose)/MinPer);
        TotalDoseToHyperCheckPoints+=Dose;
        SumDif += fabs(Dose-2.0);
        Count++;}
    /* north */
    if (i>5*RATIO)
    {
        Dose = (((Area + ySize*(i-5*RATIO) + j)->Dose)/MinPer);
        TotalDoseToHyperCheckPoints+=Dose;
        SumDif += fabs(Dose-2.0);
        Count++;}
    else //i=0
    {
        Dose = (((Area + j)->Dose)/MinPer);
        TotalDoseToHyperCheckPoints+=Dose;
        SumDif += fabs(Dose-2.0);
        Count++;}
    /* east */
    if (j+5*RATIO<ySize)
    {
        Dose = (((Area + ySize*i + (j+5*RATIO))->Dose)/MinPer);
        TotalDoseToHyperCheckPoints+=Dose;
        SumDif += fabs(Dose-2.0);
        Count++;
    }
    else //j=ySize-1
    {
        Dose = (((Area + ySize*i + (ySize-1))->Dose)/MinPer);
        TotalDoseToHyperCheckPoints+=Dose;
        SumDif += fabs(Dose-2.0);
        Count++;}
    /* south */
    if (i+5*RATIO<xSize)
    {
        Dose = (((Area + ySize*(i+5*RATIO) + j)->Dose)/MinPer);
        TotalDoseToHyperCheckPoints+=Dose;
        SumDif += fabs(Dose-2.0);
        Count++;}
    else // i = xSize-1
    {
        Dose = (((Area + ySize*(xSize-1) + j)->Dose)/MinPer);
        TotalDoseToHyperCheckPoints+=Dose;
        SumDif += fabs(Dose-2.0);
        Count++;
    }
    TempPS = TempPS -> NextPS;
}///end while static sources

// Average hyperdose value
//  A = ((float)TotalDoseToHyperCheckPoints/(float)Count)-2.f;
A = (float) SumDif/(float)(Count-1);
// Min to the tumour
B = 1.f - MinTumour/MinPer;
// Max to periphery
C = MaxPer/MinPer - 1.05f;

if (B>0.f)
    fprintf(stderr,"B bigger...\n");
    if (B > 0.f)//min in the tumour is too cold
    {
        if (C > 0.f)//too much dose leaving the tumour
            TempFloat = A+B+C;
        else        //correct dose leaving tumour
            TempFloat = (A+B);
    }// end if
    else// min in tumour OK
    {
        if (C > 0.f)//too much dose leaving the tumour
            TempFloat = (A+C);
        else        //correct dose leaving tumour
            TempFloat = (A);
    }// end else


    if (TempFloat < 0.0f)
        fprintf(stderr,"Cost -ve: A:%f B:%f C:%f Cost:%f\n",A,B,C,TempFloat);
    if (B>0.f)
        fprintf(stderr,"--- B Big ---\n");

    return(TempFloat);
}//end of function CalculateCost
```

C.2 DRand48.c

```
/*  @(#)drand48.c2.2*/
/*LINTLIBRARY*/
/*
 *  drand48, etc. pseudo-random number generator
 *  This implementation assumes unsigned short integers of at least
 *  16 bits, long integers of at least 32 bits, and ignores
 *  overflows on adding or multiplying two unsigned integers.
 *  Two's-complement representation is assumed in a few places.
 *  Some extra masking is done if unsigneds are exactly 16 bits
 *  or longs are exactly 32 bits, but so what?
 *  An assembly-language implementation would run significantly faster.
 */

//#include <stdlib.h>
#include "rand48.h"

/ srgm26feb02 addd
#include <time.h>

//srgm26feb02 added
#define DRIVER 1

ifndef HAVEFP
#define HAVEFP 1
#endif
#define N16
#define MASK((unsigned)(1 << (N - 1)) + (1 << (N - 1)) - 1)
#define LOW(x)((unsigned)(x) & MASK)
#define HIGH(x)LOW((x) >> N)
#define MUL(x, y, z){ long l = (long)(x) * (long)(y); \
        (z)[0] = LOW(l); (z)[1] = HIGH(l); }
#define CARRY(x, y)((long)(x) + (long)(y) > MASK)
#define ADDEQU(x, y, z)(z = CARRY(x, (y)), x = LOW(x + (y)))
#define X00x330E
#define X10xABCD
#define X20x1234
#define A00xE66D
#define A10xDEEC
#define A20x5
#define C0xB
#define SET3(x, x0, x1, x2)((x)[0] = (x0), (x)[1] = (x1), (x)[2] = (x2))
```

```
#define SETLOW(x, y, n) SET3(x, LOW((y)[n]), LOW((y)[(n)+1]), LOW((y)[(n)+2]))
#define SEED(x0, x1, x2) (SET3(x, x0, x1, x2), SET3(a, A0, A1, A2), c = C)
#define REST(v)for (i = 0; i < 3; i++) { xsubi[i] = x[i]; x[i] = temp[i]; } \
        return (v);
#define NEST(TYPE, f, F)TYPE f(xsubi) register unsigned short int *xsubi; { \
    register int i; register TYPE v; unsigned temp[3]; \
    for (i = 0; i < 3; i++) { temp[i] = x[i]; x[i] = LOW(xsubi[i]); }  \
    v = F(); REST(v); }
#define HI_BIT(1L << (2 * N - 1))

static void next( void );

static unsigned x[3] = { X0, X1, X2 }, a[3] = { A0, A1, A2 }, c = C;
static unsigned short lastx[3];

#if HAVEFP
double drand48( void );

double
drand48( void )
{
#if pdp11
    static double two16m; /* old pdp11 cc can't compile an expression */
    two16m = 1.0 / (1L << N); /* in "double" initializer! */
#else
    static double two16m = 1.0 / (1L << N);
#endif

    next();
    return (two16m * (two16m * (two16m * x[0] + x[1]) + x[2]));
}

//NEST(double, erand48, drand48);

#else

long irand48( register unsigned short );

long
irand48( register unsigned short m )
/* Treat x[i] as a 48-bit fraction, and multiply it by the 16-bit
 * multiplier m.  Return integer part as result.
 */
{
    unsigned r[4], p[2], carry0 = 0;

    next();
    MUL(m, x[0], &r[0]);
    MUL(m, x[2], &r[2]);
    MUL(m, x[1], p);
    if (CARRY(r[1], p[0]))
        ADDEQU(r[2], 1, carry0);
    return (r[3] + carry0 + CARRY(r[2], p[1]));
}

long
krand48( register unsigned short *xsubi, unsigned short m )
/* same as irand48, except user provides storage in xsubi[] */
{
    register int i;
    register long iv;
    unsigned temp[3];

    for (i = 0; i < 3; i++) {
        temp[i] = x[i];
        x[i] = xsubi[i];
    }
    iv = irand48(m);
    REST(iv);
}
#endif
```

```
long int
lrand48( void )
{
    next();
    return (((long)x[2] << (N - 1)) + (x[1] >> 1));
}

long int
mrand48( void )
{
    register long l;

    next();
    /* sign-extend in case length of a long > 32 bits
                      (as on Honeywell) */
    return ((l = ((long)x[2] << N) + x[1]) & HI_BIT ? l | -HI_BIT : l);
}

static void
next( void )
{
    unsigned p[2], q[2], r[2], carry0, carry1;

    MUL(a[0], x[0], p);
    ADDEQU(p[0], c, carry0);
    ADDEQU(p[1], carry0, carry1);
    MUL(a[0], x[1], q);
    ADDEQU(p[1], q[0], carry0);
    MUL(a[1], x[0], r);
    x[2] = LOW(carry0 + carry1 + CARRY(p[1], r[0]) + q[1] + r[1] +
        a[0] * x[2] + a[1] * x[1] + a[2] * x[0]);
    x[1] = LOW(p[1] + r[0]);
    x[0] = LOW(p[0]);
}

void
srand48(long int seedval)
{
    SEED(X0, LOW(seedval), HIGH(seedval));
}

unsigned short int *
seed48(unsigned short seed16v[3])
{
    SETLOW(lastx, x, 0);
    SEED(LOW(seed16v[0]), LOW(seed16v[1]), LOW(seed16v[2]));
    return (lastx);
}

void
lcong48(unsigned short int param[7])
{
    SETLOW(x, param, 0);
    SETLOW(a, param, 3);
    c = LOW(param[6]);
}

//NEST(long, nrand48, lrand48);

//NEST(long, jrand48, mrand48);

#ifdef DRIVER
/*
    This should print the sequences of integers in Tables 2
        and 1 of the TM:
    1623, 3442, 1447, 1829, 1305, ...
    657EB7255101, D72A0C966378, 5A743C062A23, ...
 */
#include <stdio.h>


// srgm09mar02 commented out to include into SA program directly now that it is checked out.
```

```
//srgm26feb02 changed from main()
/*void main(void)
{
    int i;

    // srgm26feb02 was: srand48(10);
    unsigned long int randSeed;
    time((long int*)&randSeed);
    srand48(randSeed);

    for (i = 0; i < 80; i++) {
        //srgm09mar02 tryin to get 0-1 interval was: printf("%4d ", (int)(4096 * drand48()));
        printf("%1.6f ", (float)(drand48()));
        //printf("%.4X%.4X%.4X\n", x[2], x[1], x[0]);
        // srgm09mar02 - note that x[2] IS the drand48 result!  so I can just call drand!
        printf("%f\n", (float)x[2]/65535);
    }
}
*/ // srgm09mar02 end of comment out block

endif
```

C.3 Globals.h

```
/**************************************************/
/*                                                                  */
/*  This header file contains definitions and  */
/*  variables needed in main.c for Simulated    */
/*  Annealing.                                 */
/*                                                                  */
/**************************************************/
/*                                                                  */
/*                                                                  */
/*  CREATED: 26/08/98 - present*/
/*  AUTHOR : STEVEN MILLER        */
/*  Programmed for M.Sc. Thesis*/
/*  Version: 2.6.3.a                    */
/*                                                                  */
/**************************************************/
#ifndef _GLOBALS_H
#define _GLOBALS_H 1

// conditional compile flag to use rand48 or default RNG
#define USE_RNG48 1
#define TEMP_REDUCTION_SIMPLE 0
#define MAX_TEMP_REDUCTIONS 110

#include <time.h>

// Define functions
#define sqr(x) ((x)*(x))

// Define stored values which represent different elements
#define TUMOUR 0/* the grey scale value for pgm files */
#define PERIPHERY 40/* the grey level for the periphery of the tumour */
#define POINT_SOURCE 175/* the grey scale of a point source for the pgm files */
#define EXTERNAL 255/* the grey scale value for pgm files */


// Define Debug Variables for debug compilation
#define DEBUG_MOVE_SOURCES 0/* to debug the MoveSource Alg. */
#define DEBUG_CREATE_AREA 1/* to debug CreateAreaMap fnct */
#define DEBUG_INSERT_SOURCES 0/* to debug the insert source algorithm */
#define DEBUG_CHECK_COST 0/* to debug the check cost fnct. */
#define DEBUG_INIT_TEMP 0/* to debug the initial temp. fnct */
#define DEBUG_SIM_ANN 1/* to debug the main simulated annealing loop */
#define DEBUG_COST 0/* to debug the cost function */
#define DEBUG_PLOT_DOSE 0/* to debug the 3-D dose plotting function */
```

```
#define DEBUG_CHECKHYPERDOSE 0/* to debug the checkhyperdose function */



// Default values for Simulated Annealing
#define DEFAULT_NUMBER_OF_SOURCES 1/* the default number of sources to insert into the simulation */
#define DEFAULT_DWELL_TIME 1.0/* the default amount of time a source dwells for */
#define DEFAULT_PRESCRIBED_DOSE 1.0/* the default dose for the simulation */
#define DEFAULT_INTERNAL_WEIGHT 1.0/* used to weight the contribution from internal dose */
#define DEFAULT_EXTERNAL_WEIGHT 1.0/* used to weight the contribution from external dose */
#define DEFAULT_EXTERNAL_FACTOR 0.25/* this is the percentage of prescribed dose that can be outside the
tumour*/
#define DEFAULT_TEMP_REDUCTION_FACTOR 0.9/* used to reduce the temperature of the simulation */
#define DEFAULT_MAX_ITERATIONS 200/* the number of iterations for each temperature value */
#define DEFAULT_STOP_COUNT 200/* this is the number of iterations that must have temp. within WITHIN */
#define DEFAULT_WITHIN_FACTOR 0.025/* the percentage that the cost can differ */
#define DEFAULT_TRIAL_NUMBER 1/* if no trial is provided this will be the defalut */
#define DEFAULT_MAX_INIT_TEMP_ITERATIONS 100/* the number of iterations to get the initial temp */
#define DEFUALT_INITIAL_ACCEPT_RATIO 0.95/* the acceptance ratio to start off with */
#define DEFAULT_STOP_TEMPERATURE 0.00001/* the default temperature for the simulation */
#define DEFAULT_TEMP_INCREASE_FACTOR 1.5/* used to increase the temperature in the InitTemp function */
#define DEFAULT_MOVE_PERCENT 0.98/* the probability to move a source */
#define DEFAULT_ADD_PERCENT 0.01/* the probability to add a source */
#define DEFAULT_DELETE_PERCENT 0.01/* the probability to delete a source */
#define DEFAULT_RATIO 1/* this is the ration 1 sq. == 1 mm */

#define DEFAULT_JUST_MOVE 1/* this is 1 if there is only move alowed, i.e. no insertions or deletions*/
#define DEFAULT_ALLOW_0_SOURCES 0/* if this is 1 then it is OK to have 0 sources */
#define DEFAULT_ALLOW_SOURCES_OUTSIDE 0/* if this is 1 then sources are not confined to the tumour */
#define DEFAULT_TIME_FACTOR 0/* if this is 1 then there can be more than one source at a location */
#define DEFAULT_USE_HEURISTIC 1/* usually this is set to 1 as well as JUST_MOVE so that the heuristic
method is used*/



// Define Data Structures
/* this is the structure which loads in the tumour file */
typedef struct _DoseArea{
    float Dose;/* dose at that area point */
    int Type;/* type of point (TUMOUR<EXTERNAL, etc. */
    }DoseArea;

/* this is a structuure for the point sources */
typedef struct PointSource{
    int x;  /* x position of point source */
    int y;  /* y poistion of point source */
    float DwellTime;/* dwell time of point source */
    struct PointSource *NextPS;/* pointer to next point source in the linked list */
    }PS;


// Global Variables in: "main.c"
extern DoseArea * Area;/* This is the 2D array of structures */
extern PS * FirstPS;/* This is the linked list created in InsertSources */
extern PS * FirstNewPS;/* This is the linked list created from FirstPS in MoveSources */
extern PS * FirstStaticPS;/* This is the linked list of static point sources loaded from a file */

extern float INTERNAL_WEIGHT;/* used to weight the contribution from internal dose */
extern float EXTERNAL_WEIGHT;/* used to weight the contribution from external dose */
extern float EXTERNAL_FACTOR;/* this is the percentage of prescribed dose that can be outside the
tumour*/
extern int xSize;/* This is read in from the tumour input file */
extern int ySize;/* This is read in from the tumour input file */
extern int TRIAL;
extern int NUMBER_OF_SOURCES;/* This is the number of sources in the simulation */
extern float DWELL_TIME;/* global var. for dwell_time */
extern float PRESCRIBED_DOSE;/* the prescribed dose... */
extern float TEMP_REDUCTION_FACTOR;/* the amount to decrease temp after MAX_ITERATIONS */
extern int MAX_ITERATIONS;/* the max iterations in the sim. ann. b/f temp is reduced*/
extern int STOP_COUNT;/* if this many within WITHIN_FACTOR then stop */
extern float WITHIN_FACTOR;/* this is the % of iterations that have to be within */
extern int GlobalTumourCount;/* this is to count the number of tumour spots */
extern int HyperDose;/* This is the global variable to set if the dose is too high */
extern FILE * GlobalOutputFile; /* This is where all of the output to stderr will be redirected */
extern int MAX_INIT_TEMP_ITERATIONS;/* this is the user entered number of temp. init. iterations */
extern float INITIAL_ACCEPT_RATIO;/* this is the number of accepted trials required to accept init.
temp. value */
extern float STOP_TEMPERATURE;/* this is the smallest that the temperature gets in a simulation */
extern float TEMP_INCREASE_FACTOR;
extern float MOVE_PERCENT;
extern float ADD_PERCENT;
extern float DELETE_PERCENT;
extern int RATIO;

extern int JUST_MOVE;
extern int ALLOW_0_SOURCES;
extern int ALLOW_SOURCES_OUTSIDE;
extern int TIME_FACTOR;
extern int USE_HEURISTIC;
extern char GlobalStaticFileName[128];/* For the held source filename */

/*  FUNCTION PROTOTYPES, main.c*/
int round(float x);
float distance(int x1,int y1,int x2,int y2);
void FillContoursRecursive(DoseArea *Array,int x,int y,int x_dim, int y_dim);
int PlotOutputDose(int Trial, PS **PSList);
int MakeOutputPGM(int Trial);
int CreateAreaMap(char Name[128]);
int SeedRandom(void);
int CheckCostFnct(void);
int main(int argc,char *argv[]);


/*  FUNCTION PROTOTYPES, PS.c */
void InsertSources(int NumSources);
int MoveSources(float InitialTemp,float CurTemp);
void FreePS(PS **list);
void SwapPS(PS **one, PS **two);
void CopyPS(PS **one, PS *two);
void PrintPS(PS *list);
void AddPS(PS *original_PS,PS **list);
void DeletePS(int WhichSource,PS *original_PS,PS **list);
void MakePS(PS **TempPS);
int MovePS(int WhichSource,PS *original_list,PS **new_list);
int NumPS(PS *TheList);
int AddStaticPS(char FileName[128]);


/*  FUNCTION PROTOTYPES, cost.c*/
void CheckAreaRecursive(DoseArea * Array,int x,int y,int * count);
int CheckHyperDoseSleaveIntegration(PS *ThePS);
int CheckHyperDoseSleave(PS *ThePS);
float CalculateCost(PS *TempPS);

/*  FUNCTION PROTOTYPES, sa.c*/
float GetInitialTemp(int NumSources);
int ConfigSimulatedAnnealing(char Name[128]);
float CoolingProfile(int x);
int SimulatedAnnealing(float InitialTemp, int Trial, int NumSources);

/*  FUNCTION PROTOTYPES, MyTime.c*/
void TotalTime(float Time);
void StartTime(time_t *lt);
void StopTime(time_t lt);
#endif
```

C.4 Main.c

```
/**************************************************/
/*                                                        */
/* This is the third draft of the Simulated  */
/* Annealing algorithm. It has dynamic number of*/
/* sources and can have dwell times (using con-*/
/* servation of energy.  Most parameters are */
/* set at the command line. The purpose of this */
/* version is to change the cost function so */
/* that it depends on hyper dose sleaves*/
/* and the min in the tumour.*/
/*                                                          */
/************************************************/
/*                                                          */
/*                                                          */
/*  CREATED: 12/07/1999 - present*/
/*  AUTHOR : STEVEN MILLER          */
/*  Programmed for M.Sc. Thesis*/
/*  Version: 3.0.0                          */
/*                                                          */
/************************************************/


#include <stdio.h>
#include <time.h>
#include <math.h>
#include <stdlib.h>
#include "globals.h"
#include "rand48.h"

int NUMBER_OF_SOURCES = DEFAULT_NUMBER_OF_SOURCES;/* variable for the number of sources */
float DWELL_TIME = DEFAULT_DWELL_TIME;/* global var. for dwell_time */
float PRESCRIBED_DOSE = DEFAULT_PRESCRIBED_DOSE;/* the prescribed dose... */
float INTERNAL_WEIGHT = DEFAULT_INTERNAL_WEIGHT;/* used to weight the contribution from internal dose
*/
float EXTERNAL_WEIGHT = DEFAULT_EXTERNAL_WEIGHT;/* used to weight the contribution from external dose
*/
float EXTERNAL_FACTOR = DEFAULT_EXTERNAL_FACTOR;/* this is the percentage of prescribed dose that can be
outside the tumour*/
float TEMP_REDUCTION_FACTOR = DEFAULT_TEMP_REDUCTION_FACTOR;/* this is the factor by which to decrease
the temperature */
int MAX_ITERATIONS = DEFAULT_MAX_ITERATIONS;/* the number of iterations at each temperature */
int STOP_COUNT = DEFAULT_STOP_COUNT;/* if this many within WITHIN_FACTOR then stop */
float WITHIN_FACTOR = DEFAULT_WITHIN_FACTOR;/* this is the % of iterations that have to be within */
int TRIAL = DEFAULT_TRIAL_NUMBER;/*trial number */
int MAX_INIT_TEMP_ITERATIONS = DEFAULT_MAX_INIT_TEMP_ITERATIONS;
float INITIAL_ACCEPT_RATIO = DEFUALT_INITIAL_ACCEPT_RATIO;
float STOP_TEMPERATURE = DEFAULT_STOP_TEMPERATURE;
float TEMP_INCREASE_FACTOR = DEFAULT_TEMP_INCREASE_FACTOR;
float MOVE_PERCENT = DEFAULT_MOVE_PERCENT;
float ADD_PERCENT = DEFAULT_ADD_PERCENT;
float DELETE_PERCENT = DEFAULT_DELETE_PERCENT;
int RATIO = DEFAULT_RATIO;

int JUST_MOVE = DEFAULT_JUST_MOVE;
int ALLOW_0_SOURCES = DEFAULT_ALLOW_0_SOURCES;
int ALLOW_SOURCES_OUTSIDE = DEFAULT_ALLOW_SOURCES_OUTSIDE;
int TIME_FACTOR = DEFAULT_TIME_FACTOR;
int USE_HEURISTIC = DEFAULT_USE_HEURISTIC;

/* these are the global variables */
DoseArea *Area=NULL;/* This is the 2D array of structures */
int xSize=0;/* This is read in from the tumour input file */
int ySize=0;/* This is read in from the tumour input file */
PS * FirstPS=NULL;/* This is the linked list created in InsertSources */
PS * FirstNewPS=NULL;/* This is the linked list created from FirstPS in MoveSources */
PS * FirstStaticPS=NULL;/* This is the linked list of static point sources loaded from a file */
int GlobalTumourCount = 0;/* this is to count the number of locations that the sources can go */
int HyperDose = 0;/* This is the global variable to set if there is too much dose */
int TempGlobal = 0;
FILE *GlobalOutputFile=NULL;
char GlobalStaticFileName[128];/* For the held source filename */

/*********************************************************************/
/*                  Function: Round*/
/*                              */
/*  Purpose: To round floats into integers*/
/*  Input: Any float        */
/*  Output: An integer      */
/*                              */
/*********************************************************************/
int round(float x)
{
    return((int)floor(x+0.5));// srgml1feb02 changed from ffloor
}/*end function round*/


/*********************************************************************/
/*                  Function: distance*/
/*                              */
/*  Purpose : To calculate the distance between 2 2D points*/
/*  Input : The two points (X1,Y1) and (X2,Y2) for which the dist.*/
/*  between them is required.*/
/*  Output : The distance as a float.*/
/*                              */
/*********************************************************************/
float distance(int x1,int y1,int x2,int y2)
{
    if ((x1-x2==0) && (y1-y2==0))
        return(0.0);
    else
    {
        return((float)sqrt(sqr(x1-x2)+sqr(y1-y2)));
    }
}/*end of distance function*/


/*********************************************************************/
/*                  Function FillContoursRecursive*/
/*                              */
/*  PURPOSE: This function uses recursion to "flood" a closed area in*/
/* a certain slice of the 3D array with a given value.*/
/*  INPUTS: The current position in the array, the array*/
/*  and the dimensions of the array are all inputs to the function*/
/*  OUTPUTS: None           */
/*                              */
/*********************************************************************/
void FillContoursRecursive(DoseArea *Array,int x,int y,int x_dim, int y_dim)
{

    if ((Array +x*(x_dim) + y) -> Type == EXTERNAL) /* if the position is currently "EXTERNAL" (ie:
'255') */
    {
        (Array +x*(x_dim) + y) -> Type = TUMOUR; /* fill the current position with the value */

        /* fill the area to the left, top, right, & bottom */
        if (x > 0) FillContoursRecursive(Array,x-1,y,x_dim,y_dim);
        if (y > 0) FillContoursRecursive(Array,x,y-1,x_dim,y_dim);
        if (x < x_dim-1) FillContoursRecursive(Array,x+1,y,x_dim,y_dim);
        if (y < y_dim-1) FillContoursRecursive(Array,x,y+1,x_dim,y_dim);
    }
    return;
}/*end of FillContoursRecursive*/



/*********************************************************************/
/*      Function: PlotOutputDose*/
/*                              */
/*  Purpose: This function will makd a PGM file using the  */
/*   source locations and tumour locations to find the dose */
/*  distribution.           */
/*  Inputs: Trial Number for output filename, linked list of sources*/
/*  Outputs: 1 if there is an error in the function*/
/*         else 0.                  */
/*                                  */
/**************************************************************************/
int PlotOutputDose(int Trial, PS **PSList)
{
    int i=0,j=0;/* for the for loops */
    PS *TempPS=NULL;/* Temp to run through the linked list */
    char *FileName=NULL;/* The output filename */
    FILE *OutputFile=NULL;/* the output file handle*/
    float Dose=0.0f;
    float Min=0.f;

#if DEBUG_PLOT_DOSE
    fprintf(GlobalOutputFile,"In the PlotOutputDose function\n");
#endif

    FileName = (char *)malloc(sizeof("OUTPUT/OutputDose       "));
    sprintf(FileName,"OUTPUT/OutputDose%d.dat",Trial);
    OutputFile = fopen(FileName,"w");
    if (!OutputFile)
    {
        fprintf(GlobalOutputFile,"Cannot open file %s for output, exiting PlotOutputDose\n",FileName);
        return(1);
    }
    free(FileName);

    Min = 99999.9f;
    for (i=0;i<xSize;i++)
    {
        for (j=0;j<ySize;j++)
        {
            TempPS = *PSList;
            while (TempPS)
            {
                if ((TempPS -> x == i) && (TempPS -> y == j))
                    Dose += TempPS -> DwellTime * sqr(RATIO);
                else
                    Dose += (TempPS -> DwellTime * sqr(RATIO))/sqr(distance(i,j,TempPS->x,TempPS->y));

                TempPS = TempPS -> NextPS;
            }
            if (((Area + ySize*i + j) -> Type == TUMOUR)&&(Dose<Min))
                Min = Dose;

            if (((Area + ySize*i + j) -> Type == TUMOUR))
                fprintf(OutputFile,"%f ",Dose);
            Dose = 0.0f;
        }/*end for*/
        fprintf(OutputFile,"\n");
    }/*end for*/
    fclose(OutputFile);

    /*now store the minimum in a file so that we can use it in the plotiso file*/
    FileName = (char *)malloc(sizeof("OUTPUT/MinDose       "));
    sprintf(FileName,"OUTPUT/MinDose%d.dat",Trial);
    OutputFile = fopen(FileName,"w");
    if (!OutputFile)
    {
        fprintf(GlobalOutputFile,"Cannot open file %s for output, exiting PlotOutputDose\n",FileName);
        return(1);
    }
    free(FileName);
    fprintf(OutputFile,"%f\n",Min);
    fclose(OutputFile);
    return(0);
}/*end function PlotOutputDose*/


/**************************************************************************/
/*      Function: MakeOutputPGM*/
/*                                  */
/*  Purpose: This function will make a PGM file using the  */
/*   source locations and tumour locations */
/*  Inputs: None               */
/*  Outputs: 1 if there is an error in the function*/
/*          else 0.                */
/*                                 */
/**********************************************************************/
int MakeOutputPGM(int Trial)
{
    FILE *OutputFile=NULL;
    PS *TempPS=NULL;
    int i=0,j=0;
    char *FileName=NULL;
    DoseArea *TempArea=NULL;


    FileName = (char *)malloc(sizeof("OUTPUT/SourceConfig       "));
    sprintf(FileName,"OUTPUT/SourceConfig%d.pgm",Trial);
    OutputFile = fopen(FileName,"w");
    if (!OutputFile)
    {
        fprintf(GlobalOutputFile,"Cannot open file %s for output, exiting MakeOutputPGM\n",FileName);
        return(1);
    }
    free(FileName);

    // Make the area to draw:
    if (!Area)
    {
        fprintf(GlobalOutputFile,"There is no area to draw in MakeOutputPGM, exiting!\n");
        return(1);
    }

    TempArea = (DoseArea *)malloc(xSize*ySize*sizeof(DoseArea));
    for (i=0;i<xSize;i++)
    {
        for (j=0;j<ySize;j++)
        {
            (TempArea+i*ySize+j)->Dose = (Area+i*ySize+j)->Dose;
            (TempArea+i*ySize+j)->Type = (Area+i*ySize+j)->Type;
        }
    }

    // Put in Dynamic the point sources:
    TempPS = FirstPS;
    if (!TempPS && 0)
        fprintf(GlobalOutputFile,"There are no Dynamic point sources to draw in MakeOutputPGM\n");

    while (TempPS)
    {
        (TempArea +(TempPS -> x) * ySize + TempPS -> y) -> Type = POINT_SOURCE;
        TempPS = TempPS -> NextPS;
    }

    // Put in Static the point sources:
    TempPS = FirstStaticPS;
    if (!TempPS && 0)
        fprintf(GlobalOutputFile,"There are no Static point sources to draw in MakeOutputPGM\n");

    while (TempPS)
    {
        (TempArea +(TempPS -> x) * ySize + TempPS -> y) -> Type = POINT_SOURCE;
        TempPS = TempPS -> NextPS;
    }

    fprintf(OutputFile,"P2\n");/* print file headers to the .pgm file */
    fprintf(OutputFile,"# CREATOR: XV Version 3.00  Rev: 3/30/93\n");
    fprintf(OutputFile,"%d %d\n",xSize,ySize);
    fprintf(OutputFile,"255\n");

    for (i = 0; i < xSize*ySize;i++)
    {
        if ((i % 17) == 0)/* we must print 18 points per line, hence the mod 17 */
        {
            fprintf(OutputFile,"\n%3d",(TempArea+i)->Type); /*start a new line*/
        }
        else
        {
            fprintf(OutputFile,"%4d",(TempArea+i)->Type);
        }
    }

    fclose(OutputFile);
    free(TempArea);
    return(0);
}/*end of function MakeOutputPGM*/




/************************************************************************/
/*      Function: CreatAreaMap*/
/*                                  */
/*  Purpose: This function will load and fill the tumour from file */
/*   The return value is 1 if there was an error during the */
/*   processing otherwise it is 0 */
/*  Inputs: None                */
/*                                  */
/************************************************************************/
int CreateAreaMap(char Name[128])
{
    FILE *InputFile=NULL;/*Input file pointer*/
    char name[128];/*For reading in from the input file*/
    int i=0;        /*To parse input from file*/
    int j=0;    /*To parse input from file*/
    int value=0;/*Input from the file*/


#if DEBUG_CREATE_AREA
    fprintf(GlobalOutputFile,"In CreateAreaMap\n");
#endif
    fprintf(GlobalOutputFile,"\n\n\tOpening file %s for input....\n",Name);
    if (InputFile)
        fclose(InputFile);
    if ((InputFile = fopen(Name,"r")) == NULL)
    {
        fprintf(GlobalOutputFile,"\tInput file not found exiting\n\n");
        return(1);
    }
    else
        fprintf(GlobalOutputFile,"\tFile found, opened for input.\n\n");

    /* Now parse input file to load in the tumour */
    fgets(name,128,InputFile);/* P2 */
    fgets(name,128,InputFile);/* # CREATOR: XV Version 3.01  Rev: 3/30/93 */
    /* now get the x and y dimensions of the file */
    fscanf(InputFile,"%d %d\n",&xSize,&ySize);

    fgets(name,128,InputFile);/* 255 */

    /*malloc the size of the array for the input*/
    Area = (DoseArea *)malloc(xSize*ySize*sizeof(DoseArea));

    /* now get all of the input */
    GlobalTumourCount=0;
    for (i=0;i<xSize;i++)
    {
        for (j=0;j<ySize;j++)
        {
            fscanf(InputFile,"%d",&value);
            if (ALLOW_SOURCES_OUTSIDE)
                GlobalTumourCount++;
            if (value==EXTERNAL)/*save as external location*/
            {
                (Area + i*ySize + j) -> Dose = 0.0f;
                (Area + i*ySize + j) -> Type = EXTERNAL;
            }
            else if (value == PERIPHERY)
            {
                (Area + i*ySize + j) -> Dose = 0.0f;
                (Area + i*ySize + j) -> Type = PERIPHERY;
                if (!ALLOW_SOURCES_OUTSIDE)
                    GlobalTumourCount++;
            }
            else/* it is a tumour point */
            {
                if (!ALLOW_SOURCES_OUTSIDE)
                    GlobalTumourCount++;
                (Area + i*ySize + j) -> Dose = 0.0f;
                (Area + i*ySize + j) -> Type = TUMOUR;
                GlobalTumourCount++;
            }
        }/*end for j*/
    }/*end for i*/

    /* Close the input file */
    fclose(InputFile);

#if DEBUG_CREATE_AREA
    InputFile = fopen("CreateAreaMap.pgm","w");
    fprintf(InputFile,"P2\n");
    fprintf(InputFile,"# CREATOR: XV Version 3.01  Rev: 3/30/93\n");
    fprintf(InputFile,"%i %i\n",xSize,ySize);
    fprintf(InputFile,"255\n");
    for (i=0;i<xSize;i++)
    {
        for (j=0;j<ySize;j++)
        {
            fprintf(InputFile,"%d ",(Area + i*ySize + j) -> Type);
        }
        fprintf(InputFile,"\n");
    }
    fclose(InputFile);
    fprintf(GlobalOutputFile,"Done CreateAreaMap\n");
#endif
    fprintf(GlobalOutputFile,"Done CreateAreaMap\n");
    fflush(GlobalOutputFile);
    return(0);
}/*end of CreateAreaMap*/




/***************************************************************************/
/*      Function: SeedRandom()*/
/*                                      */
/* Purpose: This function will seed the C randon number generator */
/* IT MUST ONLY BE CALLED ONCE DURING A SINGLE TIME STEP */
/* The return value is 1 if there is an error else, it is */
/* 0.                                   */
/* Input: None.                     */
/* Output: None.                */
/*                                      */
/***************************************************************************/
int SeedRandom()
{
#if USE_RNG48
    unsigned long int randSeed;// holds initial seed value

    // get a seed from the current clock time, this will always cause a differnt starting seed value
    time((long int*)&randSeed);
    // now seed the rand48 RNG
    srand48(randSeed);
    return(0);// no errors
#else
    long ltime=0;
    int stime=0;

    /* get current calender time */
    ltime = time(NULL);
    stime = (unsigned) ltime/2;
    srand(stime);
    return(0);
#endif
}/*end of SeedRandom*/





/*************************************************************************/
/*                               */
/*      Function : CheckCostFnct*/
/* Purpose: This is a function that can be called to "see" */
/* how the cost function is working. The return */
/* value is 1 if there was a problem, otherwise */
/* it is 0                     */
/* Input : None                */
/* Output : int if no problems occured, and generates a file cost.dat*/
/*                               */
/*************************************************************************/
int CheckCostFnct()
{
    FILE *CostFile=NULL;
    int i=0;
    int j=0;
    float cost=0.0f;
    float MinCost=32000.0f;
    int MinX=0,MinY=0;

    CostFile = fopen("cost.dat","w");

#if DEBUG_CHECK_COST
    fprintf(GlobalOutputFile,"In CheckCostFnct\n");
#endif

    for (i=0;i<xSize;i++)
    {
        for (j=0;j<ySize;j++)
        {
            /* if ((Area+i*ySize + j) -> Type == EXTERNAL)
            {
                fprintf(CostFile,"0.0 ");
                continue;
            } */

            FirstPS = (PS *)malloc(sizeof(PS));
            FirstPS -> x = i;
            FirstPS -> y = j;
            FirstPS -> DwellTime = DEFAULT_DWELL_TIME;
            FirstPS -> NextPS = NULL;

            cost = CalculateCost(FirstPS);
            #if DEBUG_CHECK_COST
            if (cost<MinCost)
            {
                MinCost = cost;
                MinX = i;
                MinY = j;
            }
            #endif
            fprintf(CostFile,"%f ",cost);
            FreePS(&FirstPS);
        }
        fprintf(CostFile,"\n");
#if DEBUG_CHECK_COST
    fprintf(GlobalOutputFile,"%i\n",i);
#endif
    }

    fclose(CostFile);
#if DEBUG_CHECK_COST
    fprintf(GlobalOutputFile,"Done CheckCostFnct Min of %f at x=%d and y=%d\n",MinCost,MinX,MinY);
#endif

    return(0);
}/*end of function CheckCostFnct*/



/****************************************/
/*  MAIN LINE PROGRAM           */
/****************************************/
int main(int argc,char *argv[])
{
    time_t lt;
    struct tm *ptr=NULL;
    char * FileName = NULL;

    /* setup this run */
    fprintf(stderr,"Opening file: %s for input data\n",argv[2]);
    if (ConfigSimulatedAnnealing(argv[2]))
    {
        fprintf(stderr,"Error in ConfigSimulatedAnnealing, exiting.\n");
        exit(0);
    }

    /*Open Outfile for GlobalOutputFile recording */
    FileName = (char *)malloc(sizeof("OUTPUT/Output       "));/* leave space for ## and .dat */
    sprintf(FileName,"OUTPUT/Output%d.dat",TRIAL);
    fprintf(stderr,"Opening file for recording stderr, filename:\t%s.\n\n",FileName);
    GlobalOutputFile = fopen(FileName,"w");
    free(FileName);

    fprintf(GlobalOutputFile,"\n\nThis is a program for optimizing point sources using Simulated Anneal-
ing.\n");
    fprintf(GlobalOutputFile,"Sample input is: 'SimAnn.exe 0' 'filename.pgm 1' '#sources 2' 'dwell time
3'\n 'prescribed dose 4' 'internal weight 5' 'external weight 6'\n 'percentage of prescribed to external
7' 'reduction factor 8' 'iterations 9' 'stop iterations 10' 'with in %% 11' 'trial # 12'\n\n");
    fflush(GlobalOutputFile);
    AddStaticPS(GlobalStaticFileName);

    /* loads in the tumour file(s) _and_ fills them */
    if (CreateAreaMap(argv[1]))
    {
        fprintf(GlobalOutputFile,"Error in CreateAreaMap, exiting.\n");
        exit(0);
    }

    /* Seeds the C random number generator ONLY DO THIS ONCE */
    if (SeedRandom())
    {
        fprintf(GlobalOutputFile,"Error in SeedRandom exiting.\n");
        exit(0);
    }

#if DEBUG_CHECK_COST
    /* generate 3D map of cost function */
    if (CheckCostFnct())
    {
        fprintf(GlobalOutputFile,"Error in CheckCostFnct exiting.\n");
        exit(0);
    }
```

```
#endif

    lt = time(NULL);
    ptr = localtime(&lt);
    fprintf(GlobalOutputFile,"Starting SimAnn time is: %s\n\n",asctime(ptr));
    fprintf(GlobalOutputFile,"Input file name: %s, configuration file: %s\n",argv[1],argv[2]);
    fflush(GlobalOutputFile);


    if (SimulatedAnnealing(GetInitialTemp(NUMBER_OF_SOURCES),TRIAL,NUMBER_OF_SOURCES))
    {
        fprintf(GlobalOutputFile,"Error in SimulatedAnnealing exiting.\n");
        exit(0);
    }

    lt = time(NULL);
    ptr = localtime(&lt);
    fprintf(GlobalOutputFile,"Done SimAnn time: %s\n\n",asctime(ptr));

    if (MakeOutputPGM(TRIAL))
    {
        fprintf(GlobalOutputFile,"Error in MakeOutputPGM exiting.\n");
        exit(0);
    }


    if (PlotOutputDose(TRIAL,&FirstPS))
    {
        fprintf(GlobalOutputFile,"Error in PlotOutputDose exiting.\n");
        exit(0);
    }

    /*Free global memory that was malloced*/
    free(Area);
    FreePS(&FirstPS);
    FreePS(&FirstNewPS);
    FreePS(&FirstStaticPS);
    fclose(GlobalOutputFile);
}/*end of main line*/
```

C.5 MyTime.c

```
/**************************************************/
/*                        */
/* This is a utility to calculate elapsed time*/
/* by a program.
/*                        */
/**************************************************/
/*                        */
/*                        */
/*  CREATED: 26/08/98 - present*/
/*  AUTHOR : STEVEN MILLER*/
/*  Programmed for M.Sc. Thesis*/
/*  Version: 2.6.3.a*/
/*                        */
/**************************************************/


#include <time.h>
#include <stdio.h>
#include <stdlib.h>
#include "globals.h"

typedef struct tm TM;

/*********************************************
FUNCTION NAME:TotalTime
PURPOSE:       Format the time it took for the simulation
```

```
                        into days, hours, etc.
INPUT:          The time it took the simulation
OUTPUT:         None
FUNCTIONS CALLED:fprintf
ASSUMPTIONS:None
*/
void TotalTime(float Time)
{
    int Days=0,Hours=0,Mins=0,Secs=0;


    if (Time > (60*60*24))
        Days = (int)Time/(60*60*24);

    Time -= Days*60*60*24;

    if (Time > (60*60))
        Hours = (int)Time/(60*60);

    Time -= Hours*60*60;

    if (Time > 60)
        Mins = (int)Time/60;

    Time -= Mins*60;

    Secs = (int)Time;

    fprintf(GlobalOutputFile,"Time passed is: %d-Days, %d-Hours, %d-Minutes and %d-Sec-
onds\n",Days,Hours,Mins,Secs);
}/*end function TotalTime */


/*********************************************
FUNCTION NAME:StartTime
PURPOSE:        Store the current time as the start of the simulation
INPUT:          Address of the variable to hold the start time
OUTPUT:         None
FUNCTIONS CALLED:time
                        localtime
ASSUMPTIONS:None
*/
void StartTime(time_t *lt)
{
    TM *MyTime=NULL;

    *lt = time(NULL);
    MyTime = localtime(lt);
    //fprintf(GlobalOutputFile,"Start time is: %s\n",asctime(MyTime));
}//end function StartTime


/*********************************************
FUNCTION NAME:StopTime
PURPOSE:        Store the current time as the stop time of
                        the simulation, and then calculates the total time
INPUT:          The start time
OUTPUT:         None
FUNCTIONS CALLED:fprintf
                        TotalTime
                        localtime
                        time
ASSUMPTIONS:None
*/
void StopTime(time_t lt)
{
    time_t lt1;
    float SecondsPassed=0.f;
    TM *MyTime=NULL;

    lt1 = time(NULL);
    MyTime = localtime(&lt1);
    SecondsPassed = difftime(lt1,lt);
    fprintf(GlobalOutputFile,"Stop time is: %s\n",asctime(MyTime));
    TotalTime(SecondsPassed);
    fprintf(GlobalOutputFile,"Actual processor time: %u seconds.\n",clock()/CLOCKS_PER_SEC);
}//end function StopTime
```

C.6 Ps.c

```
#include <stdlib.h>
#include <stdio.h>
#include "globals.h"
#include "rand48.h"



/********************************************
FUNCTION NAME:InsertSources
PURPOSE:        This function will insert the number of sources that
                        are used in a single solution. The sources are inserted randomly
INPUT:          The number of sources to create
OUTPUT:         None
FUNCTIONS CALLED:fprintf
ASSUMPTIONS:That the global variable exists for the point sources.
*/
void InsertSources(int num_sources)
{
    int i=0;// for loop index
    PS *TempPS=NULL;// the pointer to point sources
    PS *CurPS=NULL;// the current point source we are creating

#if DEBUG_INSERT_SOURCES
    fprintf(GlobalOutputFile,"Starting InsertSources\n");
#endif


    /* check to make sure that there are not too many sources to insert*/
    if (num_sources >= GlobalTumourCount)
    {
        fprintf(GlobalOutputFile,"There are too many sources. I have to quit there is a problem.\n");
        exit(0);
    }//end if

    // make the required number of sources
    for (i=0;i<num_sources;i++)
    {
#if DEBUG_INSERT_SOURCES
    fprintf(GlobalOutputFile,"insert source: %i\n",i);
#endif

        // call the PS function to make a point source
        MakePS(&TempPS);
        if (!FirstPS)
            FirstPS = TempPS;
        else
            CurPS -> NextPS = TempPS;
        CurPS = TempPS;
    }//end for

#if DEBUG_INSERT_SOURCES
    TempPS = FirstPS;
    fprintf(GlobalOutputFile,"In InsertSources\n");
    while (TempPS)
    {
        fprintf(GlobalOutputFile,"Pointsource x: %i  y: %i\n",TempPS -> x,TempPS -> y);
        TempPS = TempPS -> NextPS;
    }//end while
    fprintf(GlobalOutputFile,"Done insert point sources\n");
```

```
#endif
}/*end of function InsertSources*/



/*********************************************
FUNCTION NAME:MoveSourcess
PURPOSE:        This function will move sources based on the current temperature
INPUT:          The initial temperature and the current temperature
OUTPUT:         1 if there is a problem, else 0
FUNCTIONS CALLED:fprintf
ASSUMPTIONS:None
*/
int MoveSources(float InitialTemp,float CurTemp)
{
    float TempRand=0.0f;/* To hold the random number generated */
    int WhichSource=0;/* which source is going to move */
    int i=0;/* for loop counter*/
    int NumSources=0;

    // srgm11feb02 removed for windows, does not like it...
    //if (((float)((float)MOVE_PERCENT + (float)ADD_PERCENT + (float)DELETE_PERCENT) < 1.0f) ||
((float)((float)MOVE_PERCENT + (float)ADD_PERCENT + (float)DELETE_PERCENT) > 1.0f))
    //{
    //  fprintf(GlobalOutputFile,"In MoveSources and the percentages do not add up, exiting\n");
    //  return(1);
    //}

    // to get rid of the global variables for number of sources:
    NumSources = NumPS(FirstPS);

    /* this is for deciding wether to move or add */
#if USE_RNG48
    TempRand = (float)drand48();
#else
    TempRand = (float)rand()/RAND_MAX;
#endif

    if (TempRand < MOVE_PERCENT)/* this is the code for moving sources */
    {
        WhichSource=0;
#if USE_RNG48
        TempRand = (float)drand48();
#else
        TempRand = (float)rand()/RAND_MAX;
#endif
        for (i=0;i<NumSources;i++)
        {
            if ((((float)(i)/NumSources) < TempRand) && (TempRand < (float)(i+1)/NumSources))
                WhichSource = i;
        }
#if DEBUG_MOVE_SOURCES
        fprintf(GlobalOutputFile,"\nMoving %d\n",NumSources);
        if (!FirstPS)
        {
            fprintf(GlobalOutputFile,"b/f move %p\n\a",FirstPS);exit(0);
        }
        PrintPS(FirstPS);
#endif
        if (NumSources && (NumSources < GlobalTumourCount))
            MovePS(WhichSource,FirstPS,&FirstNewPS);
#if DEBUG_MOVE_SOURCES
        fprintf(GlobalOutputFile,"New points are: %p\n",FirstNewPS);
        PrintPS(FirstNewPS);
        fprintf(GlobalOutputFile,"\n");
#endif
    }/* end if move */
    else if (TempRand < MOVE_PERCENT + ADD_PERCENT)
    {
#if DEBUG_MOVE_SOURCES
        fprintf(GlobalOutputFile,"\nAdding %d\n",NumSources);
        if (!FirstPS)
        {
            fprintf(GlobalOutputFile,"b/f add %p\n\a",FirstPS);
            exit(0);
        }
        PrintPS(FirstPS);
#endif

        if (JUST_MOVE)
        {
            /* to find the lucky source to be moved */
            WhichSource=0;
#if USE_RNG48
            TempRand = (float)drand48();
#else
            TempRand = (float)rand()/RAND_MAX;
#endif
            for (i=0;i<NumSources;i++)
            {
                if ((((float)(i)/NumSources) < TempRand) && (TempRand < (float)(i+1)/NumSources))
                    WhichSource = i;
            }
            MovePS(WhichSource,FirstPS,&FirstNewPS);
        }//else if just move
        else if (NumSources < GlobalTumourCount)
            AddPS(FirstPS,&FirstNewPS);/* attach to the end of the list */

#if DEBUG_MOVE_SOURCES
    fprintf(GlobalOutputFile,"New points are: %p\n",FirstNewPS);
    PrintPS(FirstNewPS);
    fprintf(GlobalOutputFile,"\n");
#endif

    }/* end else add */
    else/* delete a source */
    {
        /* to find the lucky source to be deleted */
        WhichSource=0;
#if USE_RNG48
        TempRand = (float)drand48();
#else
        TempRand = (float)rand()/RAND_MAX;
#endif
        for (i=0;i<NumSources;i++)
        {
            if ((((float)(i)/NumSources) < TempRand) && (TempRand < (float)(i+1)/NumSources))
                WhichSource = i;
        }//end for

        //make sure there are enough sources to delete
        if (JUST_MOVE)
        {
            if (NumSources<GlobalTumourCount)
                MovePS(WhichSource,FirstPS,&FirstNewPS);
        }//end just move
        else
        {
            if (!ALLOW_0_SOURCES)
                if (NumSources>1)
                    DeletePS(WhichSource,FirstPS,&FirstNewPS);/*delete specified source */
            else
                DeletePS(WhichSource,FirstPS,&FirstNewPS);/*delete specified source */
        }//end else not just move
    }/* end of else delete*/

#if DEBUG_MOVE_SOURCES
    fprintf(GlobalOutputFile,"New Points are:\n");
    PrintPS(FirstNewPS);
    fprintf(GlobalOutputFile,"---\n");
#endif
    return(0);
}/*end of function move sources*/
```

```
/*******************************************
FUNCTION NAME:FreePS
PURPOSE:        To free a linked list of Point Sources
INPUT:          address of the first pointer
OUTPUT:         None
FUNCTIONS CALLED:none
ASSUMPTIONS:None that there was memory allocated for the list
*/
void FreePS(PS **list)
{
    PS *temp;

    temp = *list;
    while (temp)
    {
        *list = temp;
        temp = temp -> NextPS;
        free(*list);
        *list = NULL;
    }//end while
}/*end of function freePS*/


/*******************************************
FUNCTION NAME:SwapPS
PURPOSE:        This function will swap the memory of 2 INTO 1
INPUT:          The address of 2 PS linked lists
OUTPUT:         None
FUNCTIONS CALLED:None
ASSUMPTIONS:That memory has been allocated for the 2 linked lists
*/
void SwapPS(PS **one, PS **two)
{
    PS *temp=NULL;

    temp = *one;
    *one = *two;
    *two = temp;

    /* free what used to be one but is now two */
    FreePS(two);
}/*end of function swap*/


/*******************************************
FUNCTION NAME:CopyPS
PURPOSE:        This function will copy the memory of 2 INTO 1
INPUT:          The address of 2 PS linked lists
OUTPUT:         None
FUNCTIONS CALLED:None
ASSUMPTIONS:That memory has been allocated for the 2 linked lists
*/
void CopyPS(PS **one, PS *two)
{
    PS *CurPS=NULL;
    PS *CurNewPS=NULL;
    PS *NewPS=NULL;

    CurPS = two;
    while (CurPS)
    {
        NewPS = (PS*)malloc(sizeof(PS));

        NewPS -> x = CurPS -> x;
        NewPS -> y = CurPS -> y;
        NewPS -> DwellTime = CurPS -> DwellTime;
        NewPS -> NextPS = NULL;
        if (!*one)
            *one = NewPS;
        else
            CurNewPS -> NextPS = NewPS;

        CurNewPS = NewPS;
        CurPS = CurPS -> NextPS; /* go to next source */
    }//end while
}/*end of function swap*/


/*********************************************
FUNCTION NAME:PrintPS
PURPOSE:        This function will print out a linked list of PSs
INPUT:          The address of the point source list
OUTPUT:         None
FUNCTIONS CALLED:fprintf
ASSUMPTIONS:None
*/
void PrintPS(PS *list)
{
    PS *temp=NULL;
    int element=0;

    fprintf(GlobalOutputFile,"There are %d PS in the list\n",NumPS(list));
    temp = list;
    while (temp)
    {
        //fprintf(GlobalOutputFile,"Structure-%d x:%d and y:%d\n",element,temp->x,temp->y);
        fprintf(GlobalOutputFile,"%d %d\n",temp->x,temp->y);
        element++;
        temp = temp -> NextPS;
    }//end while
}/*end of function PrintPS*/



/*********************************************
FUNCTION NAME:AddPS
PURPOSE:        This function will add a point source to a list
INPUT:          The address of 2 PS linked lists
OUTPUT:         None
FUNCTIONS CALLED:None
ASSUMPTIONS:That memory has been allocated for the first linked list
*/
void AddPS(PS *original_list,PS **new_list)
{
    PS *CurPS=NULL;
    PS *CurNewPS=NULL;
    PS *NewPS=NULL;

#if DEBUG_ADDPS
    fprintf(GlobalOutputFile,"In AddPS\n");
#endif

    //need to get to the end of the list
    CurPS = original_list;
    while (CurPS)
    {
        NewPS = (PS*)malloc(sizeof(PS));

        NewPS -> x = CurPS -> x;
        NewPS -> y = CurPS -> y;
        NewPS -> DwellTime = CurPS -> DwellTime;
        NewPS -> NextPS = NULL;

        if (!*new_list)
            *new_list = NewPS;
        else
            CurNewPS -> NextPS = NewPS;
        CurNewPS = NewPS;
        CurPS = CurPS -> NextPS; /* go to next source */
    }//end while

    // now add one to the end of the list
    MakePS(&NewPS);

    // just in case there are none in the list maybe try to add it to the start
    if (!*new_list)
        *new_list = NewPS;
    else
        CurNewPS -> NextPS = NewPS;

    // store the new one
    CurNewPS = NewPS;
}/*end of function AddPS*/


/*********************************************
FUNCTION NAME:DeletePS
PURPOSE:        This function will delete a source from a list of sources
INPUT:          The address of 2 PS linked lists, and the index to the source to delete
OUTPUT:         None
FUNCTIONS CALLED:fprintf
ASSUMPTIONS:That the source to delete exists...
*/
void DeletePS(int index,PS *original_list,PS **new_list)
{
    PS *CurPS=NULL;
    PS *CurNewPS=NULL;
    PS *NewPS=NULL;
    int source_count=0;

    //fprintf(GlobalOutputFile,"look at me delete\n");
    CurPS = original_list;
    source_count=0;
    while (CurPS)
    {
        if (source_count == index) //delete this one
            CurPS = CurPS -> NextPS; /* go to next source */
        if (!CurPS)
            break;
        NewPS = (PS*)malloc(sizeof(PS));
        NewPS -> x = CurPS -> x;
        NewPS -> y = CurPS -> y;
        NewPS -> DwellTime = CurPS -> DwellTime;
        NewPS -> NextPS = NULL;

        if (!*new_list)
            *new_list = NewPS;
        else
            CurNewPS -> NextPS = NewPS;

        CurNewPS = NewPS;
        CurPS = CurPS -> NextPS; /* go to next source */
        source_count++;
    }//end while
}/*end of function DeletePS*/


/*********************************************
FUNCTION NAME:MakePS
PURPOSE:        This function will make a Point Source
INPUT:          The address of the PS to make
OUTPUT:         None
FUNCTIONS CALLED:None
ASSUMPTIONS:That memory has been allocated for the 2 linked lists
*/
void MakePS(PS **TempPS)
{
    int done=0;
    int position_OK=1;
    int time_OK=1;
    PS *CheckListPS=NULL;

    (*TempPS) = (PS *)malloc(sizeof(PS));
    while(!done)
    {
#if USE_RNG48
        (*TempPS) -> x = (round(((float)drand48())*(xSize-1)));
        (*TempPS) -> y = (round(((float)drand48())*(ySize-1)));
#else
        (*TempPS) -> x = (round(((float)rand()/RAND_MAX)*(xSize-1)));
        (*TempPS) -> y = (round(((float)rand()/RAND_MAX)*(ySize-1)));
#endif


        position_OK = 1;
        if (!ALLOW_SOURCES_OUTSIDE)
            // if position is external it is no good
            if ((Area + ((*TempPS) -> x)*ySize + (*TempPS) -> y)->Type == EXTERNAL)
                position_OK = 0;

        time_OK = 1;// assume all is well
        if (!TIME_FACTOR && position_OK) /* make sure that there are no other sources at this location*/
        {
        /* run through dynamic linked list of sources and check them all */
            CheckListPS = FirstPS;//run throught dynamic list
            while (CheckListPS)
            {
                if ((CheckListPS->x == (*TempPS)->x) && (CheckListPS->y == (*TempPS)->y))
                {
                    time_OK = 0;
                    break;
                }
                CheckListPS = CheckListPS->NextPS;
            }// end while
        /* run through static linked list of sources and check them all */
            if (time_OK)
            {
                CheckListPS = FirstStaticPS;//run through static linked list
                while (CheckListPS)
                {
                    if ((CheckListPS->x == (*TempPS)->x) && (CheckListPS->y == (*TempPS)->y))
                    {
                        time_OK = 0;
                        break;
                    }
                        CheckListPS = CheckListPS->NextPS;
                }// end while
            }//end if time_OK
        }//end if !TIME_FACTOR

        done = position_OK&&time_OK;//if position and time are OK then we are done
    }/*end while !done*/

    (*TempPS) -> DwellTime = DWELL_TIME;
    (*TempPS) -> NextPS = NULL;
}//end of MakePS


/*********************************************
FUNCTION NAME:MovePS
PURPOSE:        This function will move a point source
INPUT:          The address of 2 PS linked lists, the index of which source to move
OUTPUT:         None
FUNCTIONS CALLED:None
ASSUMPTIONS:That memory has been allocated for the 2 linked lists
*/
int MovePS(int WhichSource,PS *original_list,PS **new_list)
{
    int SourceCounter=0;/* source to move */
    int done=0;/* set to 1 if the source location is within the tumour */
    PS *TempPS=NULL;/* For NewPS linked list */
    PS *NewPS=NULL;/* For NewPS linked list */
    PS *CurNewPS=NULL;/* For NewPS linked list */
    PS *CheckListPS=NULL;/* for the check that the source has a unique location */
    float TempRand=0.0f;/* To hold the random number generated */
    int MultFactor=0;/* How far to move */
    int missed_counter=0;/* this is how many times we have not been able to move the desired source */


    TempPS = original_list;// start at the begining of the list

    MultFactor=1;
    missed_counter = 0;
    SourceCounter = 0;
    while (TempPS)
    {
        NewPS = (PS *)malloc(sizeof(PS));
        if (SourceCounter == WhichSource)
        {
            done = 0;
            while(!done)
            {
            /* MOVE IN X DIRECTION */
#if USE_RNG48
                TempRand = (float)drand48();
#else
                TempRand = (float)rand()/RAND_MAX;
#endif
                if ((TempRand < 0.333333f) && (TempPS -> x > MultFactor))
                    NewPS -> x = TempPS -> x - MultFactor;
                else if ((TempRand < 0.666666f) && (TempPS -> x < (xSize-1-MultFactor)))
                    NewPS -> x = TempPS -> x + MultFactor;
                else
                    NewPS -> x = TempPS -> x;/*TempPS -> x does not change*/

            /* MOVE IN Y DIRECTION */
#if USE_RNG48
                TempRand = (float)drand48();
#else
                TempRand = (float)rand()/RAND_MAX;
#endif

                if ((TempRand < 0.333333f) && (TempPS -> y > MultFactor))
                    NewPS -> y = TempPS -> y - MultFactor;
                else if ((TempRand < 0.666666f) && (TempPS -> y < (ySize-1-MultFactor)))
                    NewPS -> y = TempPS -> y + MultFactor;
                else
                    NewPS -> y = TempPS -> y;/*TempPS -> y does not change*/

            /* CHECK IF WE ARE DONE MOVING */
                if (ALLOW_SOURCES_OUTSIDE)// then no matter where we moved it is good
                    done = 1;
                else// make sure that it is tumour of periphery that we moved to
                {
                    if (((Area + ((NewPS) -> x)*ySize + (NewPS) -> y)->Type == TUMOUR)
                        || ((Area + ((NewPS) -> x)*ySize + (NewPS) -> y)->Type == PERIPHERY))
                        done = 1;
                    else
                    {
                        missed_counter++;
                        done = 0;
                    }
                }//end if allow_source_outside

                if (!TIME_FACTOR && done)/* then so far it is a valid location, but, lets make sure that
there are no other sources at this location*/
                {
                /* run through linked list of sources and check them all */
                    CheckListPS = original_list;
                    while (CheckListPS && done)
                    {
                        if ((CheckListPS->x == NewPS->x) && (CheckListPS->y == NewPS->y))
                        {
                            done = 0;
                            missed_counter++;
                            break;
                        }
                        CheckListPS = CheckListPS->NextPS;
                    }//end while
                /* if there was no dynamic sources there, check static ones */
                    if (done)
                    {
                /* run through linked list of sources and check them all */
                        CheckListPS = FirstStaticPS;
                        while (CheckListPS && done)
                        {
                            if ((CheckListPS->x == NewPS->x) && (CheckListPS->y == NewPS->y))
                            {
                                done = 0;
                                missed_counter++;
                                break;
                            }
                            CheckListPS = CheckListPS->NextPS;
                        }//end while
                    }
                }//end time_factor check

                /* because only one source gets moved check to make sure that it did infact move*/
                if (done && (NewPS -> x == TempPS -> x) && (NewPS -> y == TempPS -> y))
                {
                    done = 0;
                    missed_counter++;
                }

                if ((missed_counter == 25) && (!done))//then set it back to where it was and forget
about moving it
                {
                    NewPS -> x = TempPS -> x;
                    NewPS -> y = TempPS -> y;
                    done = 1;
                }
            }/* end while not done */

            //set other parameters for point source
            NewPS -> DwellTime = DWELL_TIME;
            NewPS -> NextPS = NULL;
        }/*end if SourceCounter == WhichSource*/
        else
        {
            NewPS -> x = TempPS -> x;
            NewPS -> y = TempPS -> y;
            NewPS -> DwellTime = TempPS -> DwellTime;
            NewPS -> NextPS = NULL;
        }/* end else if sourcecounter == whichsource */

        if (!(*new_list))
            (*new_list) = NewPS;
        else
            CurNewPS -> NextPS = NewPS;

        CurNewPS = NewPS;
        SourceCounter++;/* go to next source */
        TempPS = TempPS -> NextPS; /* go to next source */
    }/* end while TempPS */
    return(0);
}//end function MOvePS


/*********************************************
FUNCTION NAME:NumPS
PURPOSE:        This function return how many point sources are in the list
INPUT:          The address of a PS linked lists
OUTPUT:         The number of point source in the list
```

```
FUNCTIONS CALLED:None
ASSUMPTIONS:None
*/
int NumPS(PS *TheList)
{
    int counter=0;// counter to return
    PS *TempPS=NULL;// pointer to list of point sources

    TempPS = TheList;
    while (TempPS)
    {
        counter++;
        TempPS = TempPS -> NextPS;
    }
    return(counter);
}// end function NumPS



/*********************************************
FUNCTION NAME:AddStaticPS
PURPOSE:        This function will add NON movable sources to a list from a file
INPUT:          Filename that has the static sources in it
OUTPUT:         None
FUNCTIONS CALLED:None
ASSUMPTIONS:That the file exists
*/
int AddStaticPS(char FileName[128])
{
    FILE * m_File;
    int m_NumSources;
    int i,x,y,dt;
    PS *NewPS;
    PS *CurNewPS;
    char blah[128];

    m_File = fopen(FileName,"r");
    if (!m_File)
    {
        fprintf(GlobalOutputFile,"There is no file to open for Static sources\n");
        return(1);
    }

    // read file
    fscanf(m_File,"%d",&m_NumSources);

    // file format: x y dt
    for (i=0;i<m_NumSources;i++)
    {
        fscanf(m_File,"%d %d %d",&x,&y,&dt);
        fgets(blah,128,m_File);

        NewPS = (PS*)malloc(sizeof(PS));

        NewPS -> x = x;
        NewPS -> y = y;
        NewPS -> DwellTime = dt;
        NewPS -> NextPS = NULL;
        if (!FirstStaticPS)
            FirstStaticPS = NewPS;
        else
            CurNewPS -> NextPS = NewPS;

        CurNewPS = NewPS;
    }//end for

    return(0);
}//end function AddStaticPS
```

C.7 Rand48.h

```
// rand48.h
#ifndef _rand48_h
#define _rand48_h

/ srgm26feb02added for stand alone program
// srgm09mar02 removing for inclusion into main SA program
//void main(void);


doubledrand48 (void);
doubleerand48 (unsigned short int __xsubi[3]);
long intlrand48 (void);
long intnrand48 (unsigned short int __xsubi[3]);
long intmrand48 (void);
long intjrand48 (unsigned short int __xsubi[3]);
voidsrand48 (long int __seedval);
unsigned short int *seed48 (unsigned short int __seed16v[3]);
voidlcong48 (unsigned short int __param[7]);

#endif
```

C.8 Sa.c

```
#include <stdio.h>
#include <time.h>
#include <math.h>
#include <stdlib.h>
#include "globals.h"
#include "rand48.h"

#define ECHO 0// this is to echo the config data to the screen

/*************************************************************************/
/*                    Function: ConfigSimulatedAnnealing*/
/*                                      */
/*  Purpose : This function will initialize simulated annealing*/
/*  based on the values in the file, else the defaults.*/
/*  Inputs : FileName.     */
/*  Outputs : Integer, 0 if all is good else 1.*/
/*                                      */
/*************************************************************************/
int ConfigSimulatedAnnealing(char Name[128])
{
FILE *InputFile=NULL;/* Input file pointer */
char name[128];/* For reading in from the input file */
int Def = 0;
int TempInt=0;
int CharCount=0;
float TempFloat=0.f;


    if ((InputFile = fopen(Name,"r")) == NULL)
        Def = 1;

    if (Def)//set them all to the defaults
    {
        NUMBER_OF_SOURCES = DEFAULT_NUMBER_OF_SOURCES;
        DWELL_TIME = DEFAULT_DWELL_TIME;
        PRESCRIBED_DOSE = DEFAULT_PRESCRIBED_DOSE;
        INTERNAL_WEIGHT = DEFAULT_INTERNAL_WEIGHT;
        EXTERNAL_WEIGHT = DEFAULT_EXTERNAL_WEIGHT;
        EXTERNAL_FACTOR = DEFAULT_EXTERNAL_FACTOR;
        TEMP_REDUCTION_FACTOR = DEFAULT_TEMP_REDUCTION_FACTOR;
        MAX_ITERATIONS = DEFAULT_MAX_ITERATIONS;
        STOP_COUNT = DEFAULT_STOP_COUNT;
        WITHIN_FACTOR = DEFAULT_WITHIN_FACTOR;
        TRIAL = DEFAULT_TRIAL_NUMBER;
        MAX_INIT_TEMP_ITERATIONS = DEFAULT_MAX_INIT_TEMP_ITERATIONS;
        INITIAL_ACCEPT_RATIO = DEFUALT_INITIAL_ACCEPT_RATIO;
        STOP_TEMPERATURE = DEFAULT_STOP_TEMPERATURE;
        TEMP_INCREASE_FACTOR = DEFAULT_TEMP_INCREASE_FACTOR;
        MOVE_PERCENT = DEFAULT_MOVE_PERCENT;
        ADD_PERCENT = DEFAULT_ADD_PERCENT;
        DELETE_PERCENT = DEFAULT_DELETE_PERCENT;
        RATIO = DEFAULT_RATIO;

        JUST_MOVE = DEFAULT_JUST_MOVE;
        ALLOW_0_SOURCES = DEFAULT_ALLOW_0_SOURCES;
        ALLOW_SOURCES_OUTSIDE = DEFAULT_ALLOW_SOURCES_OUTSIDE;
        TIME_FACTOR = DEFAULT_TIME_FACTOR;
        USE_HEURISTIC = DEFAULT_USE_HEURISTIC;
    }
    else//set them to data from config. file
    {
        fscanf(InputFile,"%d\n",&TempInt);
        NUMBER_OF_SOURCES = TempInt;
        fgets(name,128,InputFile);

        fscanf(InputFile,"%f\n",&TempFloat);
        DWELL_TIME = TempFloat;
        fgets(name,128,InputFile);

        fscanf(InputFile,"%f\n",&TempFloat);
        PRESCRIBED_DOSE = TempFloat;
        fgets(name,128,InputFile);

        fscanf(InputFile,"%f\n",&TempFloat);
        INTERNAL_WEIGHT = TempFloat;
        fgets(name,128,InputFile);

        fscanf(InputFile,"%f\n",&TempFloat);
        EXTERNAL_WEIGHT = TempFloat;
        fgets(name,128,InputFile);

        fscanf(InputFile,"%f\n",&TempFloat);
        EXTERNAL_FACTOR = TempFloat;
        fgets(name,128,InputFile);

        fscanf(InputFile,"%f\n",&TempFloat);
        TEMP_REDUCTION_FACTOR = TempFloat;
        fgets(name,128,InputFile);

        fscanf(InputFile,"%d\n",&TempInt);
        MAX_ITERATIONS = TempInt;
        fgets(name,128,InputFile);

        fscanf(InputFile,"%d\n",&TempInt);
        STOP_COUNT = TempInt;
        fgets(name,128,InputFile);

        fscanf(InputFile,"%f\n",&TempFloat);
        WITHIN_FACTOR = TempFloat;
        fgets(name,128,InputFile);

        fscanf(InputFile,"%d\n",&TempInt);
        TRIAL = TempInt;
        fgets(name,128,InputFile);

        fscanf(InputFile,"%d\n",&TempInt);
        MAX_INIT_TEMP_ITERATIONS = TempInt;
        fgets(name,128,InputFile);

        fscanf(InputFile,"%f\n",&TempFloat);
        INITIAL_ACCEPT_RATIO = TempFloat;
        fgets(name,128,InputFile);

        fscanf(InputFile,"%f\n",&TempFloat);
        STOP_TEMPERATURE = TempFloat;
        fgets(name,128,InputFile);

        fscanf(InputFile,"%f\n",&TempFloat);
        TEMP_INCREASE_FACTOR = TempFloat;
        fgets(name,128,InputFile);

        fscanf(InputFile,"%f\n",&TempFloat);
        MOVE_PERCENT = TempFloat;
        fgets(name,128,InputFile);

        fscanf(InputFile,"%f\n",&TempFloat);
        ADD_PERCENT = TempFloat;
        fgets(name,128,InputFile);

        fscanf(InputFile,"%f\n",&TempFloat);
        DELETE_PERCENT = TempFloat;
        fgets(name,128,InputFile);

        fscanf(InputFile,"%d\n",&TempInt);
        RATIO = TempInt;
        fgets(name,128,InputFile);

        fscanf(InputFile,"%d\n",&TempInt);
        JUST_MOVE = TempInt;
        fgets(name,128,InputFile);

        fscanf(InputFile,"%d\n",&TempInt);
        ALLOW_0_SOURCES = TempInt;
        fgets(name,128,InputFile);

        fscanf(InputFile,"%d\n",&TempInt);
        ALLOW_SOURCES_OUTSIDE = TempInt;
        fgets(name,128,InputFile);

        fscanf(InputFile,"%d\n",&TempInt);
        TIME_FACTOR = TempInt;
        fgets(name,128,InputFile);

        fscanf(InputFile,"%d\n",&TempInt);
        USE_HEURISTIC = TempInt;
        fgets(name,128,InputFile);

        fscanf(InputFile,"%s\n",GlobalStaticFileName);
        fgets(name,128,InputFile);
    }
fclose(InputFile);

if (ECHO) //send values to the screen
{
        fprintf(stderr,"%d\n",NUMBER_OF_SOURCES);
        fprintf(stderr,"%f\n",DWELL_TIME);
        fprintf(stderr,"%f\n",PRESCRIBED_DOSE);
        fprintf(stderr,"%f\n",INTERNAL_WEIGHT);
        fprintf(stderr,"%f\n",EXTERNAL_WEIGHT);
        fprintf(stderr,"%f\n",EXTERNAL_FACTOR);
        fprintf(stderr,"%f\n",TEMP_REDUCTION_FACTOR);
        fprintf(stderr,"%d\n",MAX_ITERATIONS);
        fprintf(stderr,"%d\n",STOP_COUNT);
        fprintf(stderr,"%f\n",WITHIN_FACTOR);
        fprintf(stderr,"%d\n",TRIAL);
        fprintf(stderr,"%d\n",MAX_INIT_TEMP_ITERATIONS);
        fprintf(stderr,"%f\n",INITIAL_ACCEPT_RATIO);
        fprintf(stderr,"%f\n",STOP_TEMPERATURE);
        fprintf(stderr,"%f\n",TEMP_INCREASE_FACTOR);
        fprintf(stderr,"%f\n",MOVE_PERCENT);
        fprintf(stderr,"%f\n",ADD_PERCENT);
        fprintf(stderr,"%f\n",DELETE_PERCENT);
        fprintf(stderr,"%d\n",RATIO);

        fprintf(stderr,"%d\n",JUST_MOVE);
        fprintf(stderr,"%d\n",ALLOW_0_SOURCES);
        fprintf(stderr,"%d\n",ALLOW_SOURCES_OUTSIDE);
        fprintf(stderr,"%d\n",TIME_FACTOR);
        fprintf(stderr,"%d\n",USE_HEURISTIC);
}


return(0);
}//end function ConfigSimulatedAnnealing




/************************************************************************/
/*                  Function: CoolingProfile*/
/*                                   */
/*  Purpose : This function will return the temperature reduction factor for the current
                  iteration - based on the maximum of temperature reductions
/*  Inputs : The current temperature reduction count.*/
/*  Outputs : Float which is the temperature reduction factor.*/
/*                                   */
/************************************************************************/
float CoolingProfile(int x)
{
    float y=0.f;// this will hold the reduction factore while we build it

    // this will force the cooling profile to the sigmoid upto the max reductions,
    y = 1.0f - (1.0f/(1.0f + exp( 12.0f*(((float)MAX_TEMP_REDUCTIONS - (float)x)/
(float)MAX_TEMP_REDUCTIONS) - 5.0f)));

    // this will slide the temperature further down!
    //if (x < MAX_TEMP_REDUCTIONS)
    if (x > MAX_TEMP_REDUCTIONS)// srgm31mar02 changed
    {
        //y *= 0.9f;
        y *= TEMP_REDUCTION_FACTOR;// srgm31mar02 changed from magic number to user specified value
    }

    return y;
}//end function CoolingProfile




/************************************************************************/
/*                  Function: GetInitialTemp*/
/*                                   */
/*  Purpose : This function will return the starting temperature*/
/*  for the specific case of simulated annealing in question.*/
/*  Inputs : None.          */
/*  Outputs : Float which is the initial temperature.*/
/*                                   */
/************************************************************************/
float GetInitialTemp(int NumSources)
{
float Cost=0.0f;
float OldCost=0.0f;
float CurTemp=0.0f;
float DeltaE = 0.0f;
int M1=0;
int M2=0;
int i=0;
int PrettyOutput=0;

/* Run through a number of random seed placements and calculate the M1 M2 DeltaF values*/
/* From these values, calculate a value for InitialTemperature*/

#if DEBUG_INIT_TEMP
    fprintf(GlobalOutputFile,"GetInitTemp starting\n");
#endif
```

```
FreePS(&FirstPS);
/*Get an initial OldCost*/
InsertSources(NumSources);/* Insert N sources into the simulation */
OldCost = CalculateCost(FirstPS);/* Calculate Cost for Old cost */
FreePS(&FirstPS);

CurTemp = STOP_TEMPERATURE;/* Set the default temperature */

M1=0;
M2=0;

#if DEBUG_INIT_TEMP
    fprintf(GlobalOutputFile,"In GetInitTemp, going into main loop\n");
#endif

fprintf(GlobalOutputFile,"Heating up temperature!\n");
while (1)
{
    for (i=0;i<MAX_INIT_TEMP_ITERATIONS;i++)
    {
        InsertSources(NumSources);/*Insert N sources into the simulation*/
        Cost = CalculateCost(FirstPS);/*Calculate the cost*/
        DeltaE = Cost - OldCost;/*Find the difference*/

        if (DeltaE <= 0.0f)/*If DeltaE is negative, new cost is better so keep it*/
        {
            OldCost = Cost;
            M1++;
        }/*end if*/
#if USE_RNG48
        else if (((float)drand48()) < (exp((-DeltaE)/CurTemp)))/*else keep it with some prob.*/
#else
        else if (((float)rand()/RAND_MAX) < (exp((-DeltaE)/CurTemp)))/*else keep it with some prob.*/
#endif
        {
            OldCost = Cost;
            M2++;
        }/*end else if*/

        FreePS(&FirstPS); /*remove the point sources */
    }/*end for loop*/

    if (((float)(M1+M2)/MAX_INIT_TEMP_ITERATIONS) < INITIAL_ACCEPT_RATIO)
        CurTemp *= TEMP_INCREASE_FACTOR;
    else
    {
        fprintf(GlobalOutputFile,".\n");
        fprintf(GlobalOutputFile,"Done GetInitialTemp\n");
        fflush(GlobalOutputFile);
        return(CurTemp);
    }
    M1 = 0;
    M2 = 0;

    if (PrettyOutput/19)
    {
        fprintf(GlobalOutputFile,".\n");
        fflush(GlobalOutputFile);
        PrettyOutput=0;
    }
    else
    {
        fprintf(GlobalOutputFile,". ");
        fflush(GlobalOutputFile);
        PrettyOutput++;
    }
}/*end while*/
}/*end of function GetInitialTemp*/
```

```
/***********************************************************************/
/*          Function : SimulatedAnnealing*/
/*                                 */
/*  Purpose : This function will start cool the temperature it is */
/*  passed until it is sufficiently low that the output source*/
/*  configuration is near optimal*/
/*  Inputs : Float InitialTemp - starting temperature for the */
/*  algorithm                  */
/*  Outputs : None             */
/*                                 */
/***********************************************************************/
int SimulatedAnnealing(float InitialTemp, int Trial, int NumSources)
{
int iteration=0;/* Number of possible solutions to generate */
int done=0;/* To stop simulation */
float CurTemp=0.0f;/* The current temperature in the simulation */
float PerCheck=0.0f;/* this is the five percent value for stopping */
float Cost=0.0f;/* The cost returned from the calculate cost fnct */
float OldCost=0.0f;/* The cost of the previous solution */
float StoredOldCost=0.0f;/* this is the stored value for stop conditions */
float DeltaE=0.0f;/* The difference b/w cost and oldcost */
int m1=0,m2=0;/* temp for gathering stats on acceptance */
int StopCount=0;/* once this count gets high enough, done is true */
int IsStored=0;/* this is the flag to store the first OldCost */
long int TotalIteration=0;/* for file output of cost vesus interation */
char *FileName=NULL;/* Filename with iteration number of debug files */
FILE *CostTempFile=NULL;/* this is the cost at each temp file */
FILE *StatsFile=NULL;   /* this is a file that keeps statistics on the run */
float MaxIterations=0.f;// for estimate of number of iterations max
float AverageTime=0.f;// time calculations
time_t starttime,curtime,prevtime;// for time calculations
int Test=0;// return value from hyperdosecheck
int CurNumPS=0;// counter
PS *StoredPS=NULL;// stored for heuristic
PS *BestPSSoFar=NULL;// for stopping after resonable number of sources
float KeptCost=0.0f;
int TempReductionCount=0;// srgm17mar02 added

//int WhichSource=0,i=0;// temp ints

/* this is to calculate the number of iterations max */
#if TEMP_REDUCTION_SIMPLE//srgm31mar02 added conditional compile and #else clause
    MaxIterations = log((float)STOP_TEMPERATURE/(float)InitialTemp);
    MaxIterations = MaxIterations/(float)log(TEMP_REDUCTION_FACTOR);
    MaxIterations = MAX_ITERATIONS * MaxIterations;
#else// srgm31mar02 added else clause
    MaxIterations = log((float)STOP_TEMPERATURE/(float)(InitialTemp*CoolingPro-
file(MAX_TEMP_REDUCTIONS)));
    MaxIterations = MaxIterations/(float)log(TEMP_REDUCTION_FACTOR-.13);//srgm31mar02 added in -.13 to
account for sigmoid reduction not straight
#endif

/* report the info to output display*/
fprintf(GlobalOutputFile,"\n\nStarting SimulatedAnnealing, trial %d with:\n\tinitial temperature:
%f\n\tMax stopping temperature: %f\n",Trial,InitialTemp,STOP_TEMPERATURE);
fprintf(GlobalOutputFile,"\tThe temperature is being reduced by: %f\n",TEMP_REDUCTION_FACTOR);
fprintf(GlobalOutputFile,"\tNumber of iterations %d, within %f\n",STOP_COUNT,WITHIN_FACTOR);
fprintf(GlobalOutputFile,"\tNumber of sources: %d\n\tDwell times: %f\n\tPrescribed dose: %f\n",Num-
Sources,DWELL_TIME,PRESCRIBED_DOSE);
fprintf(GlobalOutputFile,"\t%d interations at each temperature\n",MAX_ITERATIONS);
fprintf(GlobalOutputFile,"\tWeights, internal: %f external %f\n",INTERNAL_WEIGHT,EXTERNAL_WEIGHT);
fprintf(GlobalOutputFile,"\tExternal factor: %f\n",EXTERNAL_FACTOR);
fprintf(GlobalOutputFile,"\tTotal iterations (max):%f\n",MaxIterations);

IsStored = 0;
done=0;
StopCount=0;
CurTemp = InitialTemp;
/*generate initial random solution in FirstPS*/
```

```
InsertSources(NumSources);
/*find cost of initial solution*/
OldCost = CalculateCost(FirstPS);
KeptCost = OldCost;// should not be _that_ good so keep it as a starting value

#if DEBUG_SIM_ANN
    /*Open file for 2D plot of cost and temp*/
    FileName = (char *)malloc(sizeof("OUTPUT/CostTemp       "));/* leave space for ## and .dat */
    sprintf(FileName,"OUTPUT/CostTemp%d.dat",Trial);
    fprintf(GlobalOutputFile,"In SimAnn and opening output file:\n\n\t%s \n\nfor output.\n\n",File-
Name);
    CostTempFile = fopen(FileName,"w");
    free(FileName);

    /* Open file for time projection and statistics */
    FileName = (char *)malloc(sizeof("OUTPUT/Stats       "));/* leave space for ## and .dat */
    sprintf(FileName,"OUTPUT/Stats%d.dat",Trial);
    fprintf(GlobalOutputFile,"In SimAnn and opening output file:\n\n\t%s \n\nfor output.\n\n",File-
Name);
    StatsFile = fopen(FileName,"w");
    free(FileName);
    fprintf(StatsFile,"Total Iterations (Max):%f\n",MaxIterations);
    fflush(StatsFile);
    StartTime(&starttime);
    StartTime(&prevtime);
#endif

while (!done)
{
    m1=0;
    m2=0;

    for (iteration=0;iteration<MAX_ITERATIONS;iteration++)
    {
        /* Generate solution in FirstNewPS */
        if (MoveSources(InitialTemp,CurTemp))
        {
            fprintf(GlobalOutputFile,"Error in move sources exiting SimulatedAnnealing\n");
            return(1);
        }

    /* Calculate cost */
        Cost = CalculateCost(FirstNewPS);
    /* Find energy change*/
        DeltaE = Cost - OldCost;

    /*CASE 1*/
    /* If cost is better or equale than current keep it */
        if (DeltaE <= 0.000001f)
        {
    #if DEBUG_SIM_ANN && 0
        PrintPS(FirstNewPS);
        fprintf(GlobalOutputFile,"Case 1 OldCost: %f > Cost %f\n",OldCost,Cost);
    #endif
            OldCost = Cost;
            SwapPS(&FirstPS,&FirstNewPS);
            m1++;
        }

    /*CASE 2*/
    /* else keep it with some probability */
#if USE_RNG48
        else if (((float)drand48()) < (exp((-DeltaE)/CurTemp)))/*else keep it with some prob.*/
#else
        else if (((float)rand()/RAND_MAX) < (exp((-DeltaE)/CurTemp)))/*else keep it with some prob.*/
#endif
        {
#if DEBUG_SIM_ANN && 0
        fprintf(GlobalOutputFile,"Case 2 OldCost: %f > Cost %f\n",OldCost,Cost);
#endif

            OldCost = Cost;
            SwapPS(&FirstPS,&FirstNewPS);
#if DEBUG_SIM_ANN && 0
        fprintf(GlobalOutputFile,"Rand: %f < Prob %f | at Temp: %f\n",Rand,Prob,CurTemp);
#endif
        m2++;
        }

    /*CASE 3*/
    /* else do nothing */

        else
        {
#if DEBUG_SIM_ANN && 0
            fprintf(GlobalOutputFile,"Case 3 OldCost: %f < Cost: %f\n",OldCost,Cost);
#endif
            FreePS(&FirstNewPS);
        }
        TotalIteration++;
    }/*end of for iterations...*/
    //fprintf(GlobalOutputFile,"Iterations left: %f\n",MaxIterations-(float)TotalIteration);
    /*****************************************/
    /*                    */
    /*  Calculate New Temp.*/
    /*                    */
    /*****************************************/
#if DEBUG_SIM_ANN
    /*Output data to a file to plot cooling schedule*/
    fprintf(CostTempFile,"%li %f\n",TotalIteration,OldCost);
    fflush(CostTempFile);

    /*Output stats to a file*/
    curtime = time(NULL);
    AverageTime = difftime(curtime,prevtime)/(float)MAX_ITERATIONS;
    StartTime(&prevtime);
#if TEMP_REDUCTION_SIMPLE
    fprintf(StatsFile,"(%d) Iterations left: %f Average per iteration %f, projected time remaining:
%f.\n",NumPS(FirstPS),MaxIterations-(float)TotalIteration,AverageTime,(MaxIterations-TotalItera-
tion)*AverageTime);
#else
    //fprintf(StatsFile,"(%d) Iterations left: %f Average per iteration %f, projected time remaining:
%f.\n",NumPS(FirstPS),MAX_TEMP_REDUCTIONS-(float)TempReductionCount,AverageTime,(MAX_TEMP_REDUCTIONS-TempReductionCount)*AverageTime*MAX_ITERATIONS);
// srgm31mar02 was this adding in straight cooling time
    fprintf(StatsFile,"(%d) Iterations left: %f Average per iteration %f, projected time remaining:
%f.\n",NumPS(FirstPS),(MAX_TEMP_REDUCTIONS+MaxIterations)-(float)TempReductionCount,AverageT-
ime,((MAX_TEMP_REDUCTIONS+MaxIterations)-TempReductionCount)*AverageTime*MAX_ITERATIONS);
#endif
    fflush(StatsFile);
#endif

    /*calculate new temperature*/
#if TEMP_REDUCTION_SIMPLE// srgm17mar02 modifying
        CurTemp *= TEMP_REDUCTION_FACTOR;
#else
        TempReductionCount++;
        CurTemp = InitialTemp * CoolingProfile(TempReductionCount);
        fprintf(GlobalOutputFile,"CurTemp: %f\n",CurTemp);
#endif

    #if DEBUG_SIM_ANN && 0
        fprintf(GlobalOutputFile,"The temp. is now %f and prob. of accept. is typically: %f\n",Cur-
Temp,(float)(m1+m2)/MAX_ITERATIONS);
    #endif



    /*****************************************/
    /*                    */
    /*  Check for stop conditions*/
    /*                    */
    /*****************************************/
        if (!IsStored)
        {
            IsStored = 1;
            StoredOldCost = OldCost;
        }
        PerCheck = StoredOldCost*WITHIN_FACTOR;
        if (((OldCost-PerCheck)<StoredOldCost) && (StoredOldCost<(OldCost+PerCheck)))
            StopCount++;
        else
        {
            StopCount=0;
            StoredOldCost = OldCost;
        }

        if (StopCount>STOP_COUNT)
        {
            fprintf(GlobalOutputFile,"\nFinished because stopcount reached, StopCount was: %d\n",Stop-
Count);
            fprintf(GlobalOutputFile,"Current Cost: %f\n",Cost);
            done = 1;
        }
//#if TEMP_REDUCTION_SIMPLE//srgm17mar02 added
        else if (CurTemp < STOP_TEMPERATURE)
        {
            fprintf(GlobalOutputFile,"\nFinished because CurTemp(%f)<STOP_TEMPERATURE(%f)\n",Cur-
Temp,STOP_TEMPERATURE);
            fprintf(GlobalOutputFile,"Current Cost: %f\n",Cost);
            done = 1;
        }
//#else // srgm17mar02 added
//      else if (TempReductionCount>=MAX_TEMP_REDUCTIONS)
//      {
//          fprintf(GlobalOutputFile,"\nFinished because TempReduction-
Count(%d)>=MAX_TEMP_REDUCTIONS(%d)\n",TempReductionCount,MAX_TEMP_REDUCTIONS);
//          fprintf(GlobalOutputFile,"Current Cost: %f, CurTemp: %f\n",Cost,CurTemp);
//          done = 1;
//          TempReductionCount = 0;
//      }
//#endif

        // Setup BestPSSoFar
        if (1.15*Cost<KeptCost)//best so far, therefore keep it
        {
            if (DEBUG_SIM_ANN)
                fprintf(GlobalOutputFile,"Keep new config as best so far, it is better %f than
%f\n",Cost,KeptCost);
            KeptCost = Cost;
            FreePS(&BestPSSoFar);
            CopyPS(&BestPSSoFar,FirstPS);
            MakeOutputPGM(666);
        }

        // Check if there are too many sources to keep going
        if (NumPS(FirstPS) == 15)// too many sources, so quit
        {
            // set output to best so far
            FreePS(&FirstPS);
            CopyPS(&FirstPS,BestPSSoFar);
            fprintf(GlobalOutputFile,"Reached limit of number of sources (15), terminating with best so
far.\n");
            break;
        }

    /****************************************/
    /*                  */
    /*  USE HEURISTIC*/
    /*                  */
    /****************************************/
    if (USE_HEURISTIC && done)
    {
        // print out the current config so that we can see how it did
        MakeOutputPGM(Trial++);

        // first time through store FirstPS right away so that it can be used if all is OK
        if (!StoredPS)
            CopyPS(&StoredPS,FirstPS);

        Test = CheckHyperDoseSleave(FirstPS);
        CheckHyperDoseSleaveIntegration(FirstPS);
        if (Test == 1)//too much dose at check points SO ADD
        {
            CurNumPS = NumPS(FirstPS);
            if (CurNumPS < GlobalTumourCount)
            {
                fprintf(GlobalOutputFile,"Hyperdose sleave is too big so adding (from %d --> %d)\n",Cur-
NumPS,CurNumPS+1);
            // store the current config in case it is better
                FreePS(&StoredPS);//In case there is already a StoredPS
                SwapPS(&StoredPS,&FirstPS);
            // init code for SA
                IsStored = 0;
                done=0;
                StopCount=0;
                TotalIteration=0;
                TempReductionCount=0;// srgm06apr02 added
                CurTemp = GetInitialTemp(CurNumPS++);
            /* this is to calculate the number of iterations max */
#if TEMP_REDUCTION_SIMPLE// srgm31mar02 adding in else clause
                MaxIterations = log((float)STOP_TEMPERATURE/(float)CurTemp);
                MaxIterations = MaxIterations/(float)log(TEMP_REDUCTION_FACTOR);
                MaxIterations = MAX_ITERATIONS * MaxIterations;
#else// srgm31mar02 added else clause
                MaxIterations = log((float)STOP_TEMPERATURE/(float)(InitialTemp*CoolingPro-
file(MAX_TEMP_REDUCTIONS)));
                MaxIterations = MaxIterations/(float)log(TEMP_REDUCTION_FACTOR-.13);//srgm31mar02 added
in -.13 to account for sigmoid reduction not straight
#endif
            /*generate initial random solution in FirstPS*/
                InsertSources(CurNumPS++);
            /*find cost of initial solution*/
                OldCost = CalculateCost(FirstPS);
            }
            else
            {
                fprintf(GlobalOutputFile,"There are too many sources, yet I NEED more!\n");
                exit(0);
            }
        }
        else if (Test == -1)   // not enough dose
        {
            fprintf(GlobalOutputFile,"Hyperdose too small going with previous as answer\n");
            // restore the previous answer;
            FreePS(&FirstPS);// in case there is already a FirstPS
            SwapPS(&FirstPS,&StoredPS);
        }
        else
        {
            fprintf(GlobalOutputFile,"Exactly correct dose configuration, done SimAnn\n");
        }
    }// end if use_heuristic && done

        fflush(GlobalOutputFile);
}/*end of while*/

#if DEBUG_SIM_ANN
    fclose(CostTempFile);
    fclose(StatsFile);
    StopTime(starttime);
#endif

fprintf(GlobalOutputFile,"After SimulatedAnnealing Source Locations are:\n");
PrintPS(FirstPS);
return(0);
}/*end of function SimulatedAnnealing*/
```