Vacation Scholarship Report

Thomas Taimre

21 January, 2004

1 Outline of Work Undertaken

I began the scholarship by acquainting myself with the theory of the Cross-Entropy method, and writing MATLAB programs which applied this method to simple problems.
After this introductory period, I began work on the 8 Queens problem. The challenge was to use the Cross-Entropy (CE) method to find ways to place 8 queens on an 8x8 chessboard in such a way that none of the queens threaten any other queen. The algorithms implementing CE for this evolved over the course of the scholarship, eventually making use of FACE (the Fully Adaptive CE algorithm).
After I had progressed somewhat with the Queens problem, I began to look at the Travelling Salesman Problem (TSP), and related problems. I reviewed the algorithms, and implemented them in MATLAB. Once I had done this I looked for some test cases for these on the Internet. The Permutation Flow-Shop Problem is very similar to the TSP, and so I looked at this as a minor modification, again creating some MATLAB code.
Towards the middle of the scholarship, I looked at a few more example problems described in [2], and generated more MATLAB code for the Bayesian image reconstruction case.
After this was finished, I began to look at the problem of Clustering, with the aid of the CE algorithm. Two ways of attacking clustering problems with CE are described in [2]; a discrete and a continuous method. I implemented them both, and then ran several extensive experiments on well known test data sets. After spending much time modifying the clustering programs, and testing them on test data, I spent the last few days looking at ways to visualise the CE algorithm, and implementing some interesting modifications to the standard CE method. As a result, I created a few small example programs in MATLAB, as well as modifying existing programs in order to see how the modifications behaved.

1.1 Summary

Throughout the scholarship, I have sought to improve my understanding of the Cross-Entropy method and its various applications, both by reading and implementation. I have also improved my programming skills by application of the theory to numerous optimisation problems.

2 Programs

In this section, I list many of the MATLAB programs I have written during the course of this scholarship, along with descriptions of their function, and instructions on their use.

2.1 Illustrative Example Problems

Problem Description

This problem is described in Example 5.1 in [2].

Program Descriptions

`toyexample.m`
: This program runs an interactive demonstration of the convergence of the CE method using Normal updating in 1 dimension. Click the mouse or press a key to advance the display by one iteration.

Usage:
: Call the program from MATLAB, with the following syntax:

```
toyexample
```

`toyexample2.m`
: This program runs an alternate interactive demonstration of the convergence of the CE method using Normal updating in 1 dimension. Click the mouse or press a key to advance the display by one iteration.

Usage:
: Call the program from MATLAB, with the following syntax:

```
toyexample2
```

`ssmall.m`
: This program illustrates the evolution of the CE method on a very simple problem, showing that it is quite possible to have the standard deviation parameter reduced below an extremely small number.

Usage:
: Call the program from MATLAB, with the following syntax:

σ=`ssmall(`N`,`ϱ`)`

Example: sig=ssmall(200,0.1)

Inputs:
- N - number of samples each iteration
- ϱ - fraction of best performing samples to take

Outputs:
- σ - A vector of all of the std. deviations

2.2 Bayesian Image Reconstruction

Problem Description

This problem is described in Exercise 8.4.5 in [2]

Program Descriptions

`bayes.m`

This program reconstructs the image in the first problem via the CE method.

Usage:

Call the program from MATLAB, with the following syntax:

`[`x`,`y`]=bayes(`N`,`ϱ`,`α`,`σ`,`y`)`

Example: [reimage,oldimage]=bayes(500,0.04,0.7,0.1,image)
In this example, image is a matrix, consisting of two unique gray levels (eg. 0's and 1's), and the noise to be added is distributed Normally, with a standard deviation of 0.1.

Inputs:
- N - number of samples each iteration
- ϱ - fraction of best performing samples to take
- α - smoothing paramter
- σ - std. deviation for image noise (optional)
- y - image data (optional)

Outputs:
- x - reconstructed image
- y - original image

`bayes2.m`

This program reconstructs the image in the second problem via the CE method.

Usage:
Call the program from MATLAB, with the following syntax:

`[`x`,`y`,`v`]=bayes2(`N`,`ϱ`,`α`,`σ`,`y`)`

Example: [reimage,oldimage,probs]=bayes2(500,0.04,0.7)
In this example, a default image is used.

Inputs:

N	-	number of samples each iteration
ϱ	-	fraction of best performing samples to take
α	-	smoothing paramter
σ	-	std. deviation for image noise
y	-	image data

Outputs:

x	-	reconstructed image
y	-	original image
v	-	probabilities at the end

`scoreb.m` This program is used internally by both of the above programs to evaluate the performance of the algorithm.

2.3 The Permutation Flow-Shop Problem

Problem Description

This problem is described in Exercise 7.6.4 in [2].

Program Descriptions

`pfsp.m`
This MATLAB program gives the best found permutation via the CE method.

Usage:
Call the program from MATLAB, with the following syntax:

`[`π`,`t`]=pfsp(`N`,`ϱ`,`α`,`$traj$`,`n`,`m`,`t`,`z`)`

Example: [perm,cost]=pfsp(100,0.1,0.8,2,10,2,t,10)
where t is a 10 x 2 matrix of costs.

Inputs:

N	-	Number of samples to generate each round
ϱ	-	fraction of best samples to take
α	-	smoothing parameter
$traj$	-	0, node transitions, x(1)=1
		1, node transitions, x(1) random
		2, node placement (default)
n	-	number of jobs
m	-	number of machines
t	-	$t(i,j)$ is the cost of job i on machine j
z	-	number of succesive rho-th quantiles the same before stop

Outputs:

π	-	the output permutation
t	-	the cost (in time) matrix used

`gpfsp.m` This program is used internally to generate tours via node transitions.

`gpfsp2.m` This program is used internally to generate tours via node placement, using the first row of the P matrix to generate the first position.

`gpfsp3.m` This program is used internally to generate tours via node placement, using a random row of the P matrix to generate the first position.

`fpfsp.m`

This MATLAB program gives the best found permutation via the CE method, using "fast" trajectory generation, as mentioned in Remark 4.12, and described in Algorithms 4.11.3 and 4.11.4, as well as Section 4.11.2 in [2].

Usage:

Call the program from MATLAB, with the following syntax:

`[`π`,`t`]=fpfsp(`N`,`ϱ`,`α`,`$traj$`,`n`,`m`,`t`,`z`)`

Example: [perm,cost]=fpfsp(200,0.05,0.75,3,12,1,t,10)
where t is a 12 x 3 matrix of costs.

Inputs:

N	-	Number of samples to generate each round
ϱ	-	fraction of best samples to take
α	-	smoothing parameter
$traj$	-	0, alias technique (default) uses x% = 70%
		1, composition method
n	-	number of jobs
m	-	number of machines
t	-	$t(i,j)$ is the cost of job i on machine j
z	-	number of succesive rho-th quantiles the same before stop

Outputs:

π	-	the output permutation
t	-	the cost (in time) matrix used

`fgpfsp.m` This program is used internally to generate tours via the alias speed-up.

`fgpfsp2.m` This program is used internally to generate tours via the composition speed-up.

`spfsp.m` This program is used internally to evaluate the performance of these algorithms.

2.4 Travelling Salesman Problem

Problem Description

This problem is described in Section 4.7 of [2].

Program Descriptions

`tsp.m`
This MATLAB program gives the best found tour via the CE method.

Usage:
Call the program from MATLAB, with the following syntax:

π=`tsp(`N`,`ϱ`,`α`,`A`,`$traj$`)`

Example: tour=tsp(1000,0.05,0.8,A,0)
where A is a matrix of lengths between nodes.

Inputs:

N	-	Number of samples to generate each round
ϱ	-	fraction of best samples to take
α	-	smoothing parameter
A	-	$A(i,j)$ is the distance between node i and node j
$traj$	-	0, node placements 1, node transitions

Outputs:

π	-	the best tour found

`gtsp0.m` This program is used internally to generate tours via node placements.

`gtsp1.m` This program is used internally to generate tours via node transitions.

`stsp.m` This program is used internally to evaluate the performance of a particular tour.

`minitsp.m` This is a program which generates a smaller TSP problem (with a specified number of nodes) from a larger one. This idea is mentioned in Remark 4.13 in [2].

2.5 The n Queen Problem

Problem Description

This problem is described in Excercise 2.6.6 in [2].

Program Descriptions

`q.m`

This MATLAB program gives the best found placement for n queens on an n x n chessboard using the FACE algorithm.

Usage:

Call the program from MATLAB, with the following syntax:

B=`q`$(N_e, N_{min}, N_{max}, \alpha, d, c, n)$

Example: board=q(15,300,2000,0.7,10,5,8)

Inputs:

N_e - Number of elite samples to use
N_{min} - Minimum number of samples to use (must be $\geqslant$ Ne)
N_{max} - Maximum number of samples to use (must be $\geqslant$ Ne)
α - Smoothing Parameter
d - number of S_t^* the same in a row with no $\widehat{\gamma}_t$ improvement
c - number of $N_t = N_{max}$ in a row
n - n queens on an nxn board

Outputs:

B - An n x n matrix with queens denoted by 1s, and blank squares denoted by 0s

`genq.m` This program is used internally to generate chessboard outcomes.

`scoreq.m` This program is used internally to evaluate the performance of a particular board.

`wq.m`
This program takes in a set of exact solutions to the 8x8 queen problem, and then tells you which of the 12 unique (disregarding reflections and rotations) solutions you have found. The exact solutions were found in [1].

Usage:
Call the program from MATLAB, with the following syntax:

v=`wq(`U`)`

Inputs:

U - An 8 x 8 x k matrix of exact solutions

Outputs:

v - A vector of length k, labelling the solutions found

2.6 Clustering

Problem Description

Clustering problems are described in Section 8.3 of [2].

Program Descriptions

`NCE.m`
This program finds a set of cluster means via the CE method with (independent) Normal updating.

Usage:

Call the program from MATLAB, with the following syntax:

`[`μ`,`$count$`,`$score$`]=NCE(`N`,`g`,`α`,`k`,`$data$`,`$modif$`,`$drplot$`,`c`,`σ_0`)`

Example: [mu,count,score]=NCE(1000,10,0.7,5,DATA,0,1)

Inputs:

N	-	Number of samples to generate each iteration
g	-	Number of these samples to use to update parameters
α	-	Smoothing parameter
k	-	Number of cluster means to find
$data$	-	The data we are trying to fit means to (Should be n x d, where there are n points, and d dimensions)
$modif$	-	If 1, use modified smoothing, otherwise use standard smoothing
$drplot$	-	If 1, draws the cluster means and the data (for 2-dimensions)
c	-	Optional starting centroids
σ_0	-	Optional starting standard deviation

Outputs:

μ	-	The centroids found via the CE method, using Normal updating with the parameter set
$count$	-	The number of iterations taken
$score$	-	The final score of these centroids

`genNCE.m` This program is used internally to generate the cluster means.

`scoreNCE.m` This program is used internally to evaluate the performance of a particular set of means against the data.

`MCE.m`

This program finds a set of cluster means via the CE, looking at clustering as a "mincut" type problem.

Usage:

Call the program from MATLAB, with the following syntax:

x`=MCE(`N`,`ϱ`,`α`,`k`,`$data$`)`

Example: x=MCE(2000,0.01,0.6,3,DATA)

Inputs:

N	-	Number of samples to generate each iteration
ϱ	-	The fraction of samples used to update the probabilities
α	-	Smoothing parameter
k	-	Number of clusters to assign points to
$data$	-	The data we are trying to assign to clusters (Should be n x d, where there are n points, and d dimensions)

Outputs:

x	-	The best found assignment of the data points

`genMCE.m` This program is used internally to assign data points to clusters.

`MCEJ.m`
This is slight modification of the above program, using the "injection" idea (due to Zdravko Botev). It generally produces superior results to the unmodified MCE method.

Usage:
Call the program from MATLAB, with the following syntax:

x=`MCEJ(`N`,`ϱ`,`α`,`k`,`$data$`)`

Example: x=MCEJ(1400,0.03,0.75,4,DATA)

Inputs:

N	-	Number of samples to generate each iteration
ϱ	-	The fraction of samples used to update the probabilities
α	-	Smoothing parameter
k	-	Number of clusters to assign points to
$data$	-	The data we are trying to assign to clusters (Should be n x d, where there are n points, and d dimensions)

Outputs:

x	-	The best found assignment of the data points

`scoreMCE.m` This program is used internally to evaluate the performance of a particular assignment against the data.

`clNCE.m` This is a small program which labels the points in a dataset according to a set of cluster means.

Usage:
Call the program from MATLAB, with the following syntax:

x=`clNCE(`c`,`$data$`,`k`)`

Example: x=clNCE(mu,DATA,5)

Inputs:

c	-	A set of cluster means
data	-	The data we are trying to assign to clusters (Should be n x d, where there are n points, and d dimensions)
k	-	Number of clusters to assign points to

Outputs:

x	-	The assignment of the data points to clusters

`cMCE.m` This is a small program which calculates cluster means from a given cluster labelling.

Usage:
Call the program from MATLAB, with the following syntax:

`c=cMCE(`x`,`y`,`k`)`

Example: mu=cMCE(x,data,6)

Inputs:

x	-	A labelling of data points
data	-	The data we are trying to fit means to (Should be n x d, where there are n points, and d dimensions)
k	-	Number of cluster means

Outputs:

x	-	The cluster means calculated for this labelling of data points

2.7 The Maze Problem

Problem Description

This problem is looked at in Section 8.2.2 of [2]. Section 8.2 discusses using CE more generally in this way.

Program Descriptions

`maze.m`
This program tries to find the shortest path through a maze, by generating a set of choices at junctions. For example, an output may indicate something along the lines of "At Junction 1, go East, at the next Junction, go South ...". The set of choices is updated via the CE method.

Usage:

Call the program from MATLAB, with the following syntax:

π=maze($N, \varrho, \alpha, M, start, finish$)

Example: pi=maze(1000,0.02,0.7,MAZE,[1,1],[12,12])
Where, in this example, M is a 12x12 maze of 0s and 1s, with 1s representing walls, and 0s representing the paths. The start of the maze is at [1,1] and the end is at [12,12]. Note that this method seems to perform quite well on very small mazes, or on mazes with few junctions, but performs less well on larger problems.

Inputs:

N	-	Number of samples to generate each iteration
ϱ	-	Fraction of samples to use to update the choices
α	-	Smoothing parameter
M	-	A matrix of 1s and 0s, representing a maze
$start$	-	Starting position
$finish$	-	Ending position

Outputs:

π	-	Output choice set

`gmaze.m` This program is used internally to generate a set of direction choices.

`smaze.m` This program is used internally to evaluate the performance of a given set of direction choices.

2.8 Rosenbrock Visualisation

Problem Description

This tricky function is described in Section 5.1 of [2].

Program Descriptions

`demoros.m`
This program contains a set of default parameters, visually illustrating the convergence of the CE method using Normal updating with "modified smoothing" (Remark 5.2 in [2]) for the 2-dimensional Rosenbrock function. The base programs were written by Sho Nariai, and subsequently modified.

Usage:

Call the program from MATLAB, with the following syntax:

```
demoros
```

`ceros.m` This program uses the CE method to update the parameters.

`rosenbrock.m` This program is used internally to evaluate the value of the Rosenbrock function at a particular point.

`tinormrand.m` This program is used internally to generate sample points for evaluation.

`plotce.m` This program is used internally to plot the progress of the mean and covariance in 2-dimensions.

References

[1] V. Chvatal. All solutions to the problem of eight queens. http://www.cs.rutgers.edu/~chvatal/8queens.html.

[2] D. P. Kroese and R. Y. Rubinstein. *The Cross-Entropy Method: A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation and Machine Learning.* Springer, 2004.