Proceedings of the 2007 Winter Simulation Conference
S. G. Henderson, B. Biller, M.-H. Hsieh, J. Shortle, J. D. Tew, and R. R. Barton, eds.

PARALLEL CROSS-ENTROPY OPTIMIZATION

Gareth E. Evans

Department of Mathematics
University of Queensland
Brisbane, QLD 4072, AUSTRALIA

Jonathan M. Keith

School of Mathematical Sciences
Queensland University of Technology
Brisbane, QLD 4001, AUSTRALIA

Dirk P. Kroese

Department of Mathematics
University of Queensland
Brisbane, QLD 4072, AUSTRALIA

ABSTRACT

The Cross-Entropy (CE) method is a modern and effective optimization method well suited to parallel implementations. There is a vast array of problems today, some of which are highly complex and can take weeks or even longer to solve using current optimization techniques. This paper presents a general method for designing parallel CE algorithms for Multiple Instruction Multiple Data (MIMD) distributed memory machines using the Message Passing Interface (MPI) library routines. We provide examples of its performance for two well-known test-cases: the (discrete) Max-Cut problem and (continuous) Rosenbrock problem. Speedup factors and a comparison to sequential CE methods are reported.

1 INTRODUCTION

1.1 The Cross Entropy Method

The *Cross Entropy* method (Rubinstein and Kroese 2004) or CE method can be used for two types of problems:

- Estimation,
- Optimization.

Here we focus on parallel implementation of the CE method for optimization problems, although similar techniques will also be possible for estimation. Suppose we wish to solve the following maximization problem: Let $\mathscr{X}$ be a finite set of *states* and S a real-valued *performance function* on $\mathscr{X}$. We wish to find the maximum value of S over $\mathscr{X}$ and the state(s) corresponding to this value. Let γ^* be the maximum of S over $\mathscr{X}$ and let $\mathbf{x}^*$ be a state at which this maximum is attained. Then,

$$S(\mathbf{x}^*) = \gamma^* = \max_{\mathbf{x}\in\mathscr{X}} S(\mathbf{x}). \tag{1}$$

The CE method is an iterative optimization method that starts with a parameterized sampling distribution $f(\mathbf{x};\mathbf{u})$ from which a random sample is generated. Each observation in this sample is *scored* for its performance as the solution to a specified optimization problem. A fixed number of the best of these observations are referred to as the *elite sample*. This elite sample is used to update the parameters for the sampling distribution. A new sample and elite sample are then generated from the updating sampling distribution. The sampling distribution eventually converges to a degenerate distribution about the final locally optimal solution which ideally will be globally optimal.

The first step of the CE method is to turn the optimization problem (1) into a meaningful *estimation problem*. Let $I_{\{S(\mathbf{X})\geq\gamma\}}$ be a collection of indicator functions for various levels γ. Then for the discrete case we associate the estimation of

$$\ell(\gamma) = \mathbb{P}_{\mathbf{u}}(S(\mathbf{X}) \geq \gamma) = \sum_{\mathbf{x}} I_{\{S(\mathbf{x})\geq\gamma\}} f(\mathbf{x};\mathbf{u}) = \mathbb{E}_{\mathbf{u}}\left[I_{\{S(\mathbf{X})\geq\gamma\}}\right]$$

with (1). Now we use a two-part iterative approach to obtain $\hat{\gamma}_1, \hat{\gamma}_2, \ldots, \hat{\gamma}_i$ and corresponding parameter vectors $\hat{\mathbf{v}}_1, \hat{\mathbf{v}}_2, \ldots, \hat{\mathbf{v}}_i$ such that $\hat{\gamma}_i \to \gamma^*$ and $f(\mathbf{x};\hat{\mathbf{v}}_i)$ approaches the degenerate distribution about $\mathbf{x}^*$. Let ρ be a real number between 0 and 1 representing the proportion of the sample taken as the elite sample. For a random sample $\mathbf{X}_1, \ldots, \mathbf{X}_N$ let $S_{(1)} \leq \ldots \leq S_{(N)}$ be the performances of $S(\mathbf{X}_i)$ ordered

from smallest to largest. Thus, $S_{(j)}$ is called the j-th *order-statistic* of the sequence $S(\mathbf{X}_1),\ldots,S(\mathbf{X}_N)$.

For fixed $\hat{\gamma}_t$ and $\hat{\mathbf{v}}_{t-1}$, derive $\hat{\mathbf{v}}_t$ from the solution of the following program

$$\max_{\mathbf{v}} D(\mathbf{v}) := \max_{\mathbf{v}} \frac{1}{N}\sum_{i=1}^{N} I_{\{S(\mathbf{X}_i)\geq\hat{\gamma}_t\}} \ln f(\mathbf{X}_i;\mathbf{v}) \qquad (2)$$

Algorithm 1.1. *[CE Algorithm for Optimization]*

1. *Choose an initial parameter vector* $\hat{\mathbf{v}}_0$. *Set* $t = 1$.
2. *Generate a sample* $\mathbf{X}_1,\mathbf{X}_2,\ldots,\mathbf{X}_N$ *from the density* $f(\cdot;\hat{\mathbf{v}}_{t-1})$ *and compute the sample* $(1-\rho)$*-quantile* $\hat{\gamma}_t$ *of the performance according to* $\hat{\gamma}_t = S_{(\lceil(1-\rho)N\rceil)}$.
3. *Using the same sample* $\mathbf{X}_1,\mathbf{X}_2,\ldots,\mathbf{X}_N$ *solve the stochastic program (2) and denote the solution* $\hat{\mathbf{v}}_t$.
4. *If for some* $t \geq d$, *say* $d = 5$,

$$\hat{\gamma}_t = \hat{\gamma}_{t-1} = \cdots = \hat{\gamma}_{t-d}, \qquad (3)$$

then stop, otherwise set $t = t+1$ *and iterate from Step 2.*

1.2 The Max-Cut Problem

Given a weighted undirected graph $G = (V,E)$, where $V = \{1,\ldots,n\}$ is the set of vertices and E is the set of edges with associated weights c_{ij} between vertices i and j, the Max-Cut problem is: what partition of the vertices into two distinct subsets V_1 and V_2 maximizes the sum of the weights of the edges e_{ij} where vertices i and j are in different subsets? We can assume without loss of generality that the graph G is complete and weights c_{ij} are non-negative. It can be shown that the Max-Cut problem is NP-Complete (Karp 1972).

We can represent the edge weights via a non-negative, symmetric cost matrix $C = (c_{ij})$ where c_{ij} is the weight of the edge between vertices i and j. The cost of a cut (its score) is then the sum of the weights of the edges with vertices i and j in different partitions. For example, if we had the cut $\{\{1,3\},\{2,4\}\}$ with the following cost matrix

$$\begin{pmatrix} 0 & c_{12} & 0 & c_{14} \\ c_{21} & 0 & c_{23} & c_{24} \\ 0 & c_{32} & 0 & 0 \\ c_{41} & c_{42} & 0 & 0 \end{pmatrix}$$

the cost of the cut would be $c_{12}+c_{14}+c_{23}$.

We represent a cut as a vector $\mathbf{x} = (x_2,\ldots,x_n)$ where $x_i = 1$ if node i is in the same partition as node 1, and $x_i = 0$ otherwise. To generate our samples we let $X_2,\ldots,X_n$ be independent Bernoulli random variables with success probabilities $p_2,\ldots,p_n$. The solution to the stochastic program (2) which is used to update the sampling distribution in algorithm 1.1 now becomes

$$\hat{p}_{t,i} = \frac{\sum_{k=1}^{N} I_{\{S(\mathbf{X}_k)\geq\hat{\gamma}\}} I_{\{X_{ki}=1\}}}{\sum_{k=1}^{N} I_{\{S(\mathbf{X}_k)\geq\hat{\gamma}\}}}. \qquad (4)$$

For our Max-Cut experiments we construct an artificial network such that the optimal solution is known. To construct this network on n nodes for $m \in \{1,\ldots,n\}$ let

$$C = \begin{pmatrix} Z_{11} & B_{12} \\ B_{21} & Z_{22} \end{pmatrix},$$

where Z_{11} is an $m \times m$ symmetric matrix with all the upper-diagonal elements generated from a $\mathsf{U}(0,1)$ distribution. Z_{22} is a $(n-m)\times(n-m)$ symmetric matrix generated in the same way as Z_{11}. All other elements are 1 apart from the diagonal elements which are 0. The optimal cut is given by $\mathbf{V}^* = \{\mathbf{V}_1^*,\mathbf{V}_2^*\}$ with

$$\mathbf{V}_1^* = \{1,\ldots,m\} \text{ and } \mathbf{V}_2^* = \{m+1,\ldots,n\},$$

while the optimal cut value is

$$\gamma^* = m(n-m).$$

1.3 The Rosenbrock Problem

The n-dimension Rosenbrock function is

$$S(x) = \sum_{i=1}^{n-1} 100(x_{i+1}-x_i^2)^2 + (x_i-1)^2. \qquad (5)$$

To minimize (5) via the CE method we generate a random vector $\mathbf{X} = (X_1,X_2,\ldots,X_n)$ where each component X_i is generated independently from a normal distribution with parameters $\hat{\mu}_{t-1}$ and $\hat{\sigma}^2_{t-1}$. Determine $\hat{\gamma}_t = S_{(\lceil(1-\rho)N\rceil)}$ and update $\hat{\mu}_t$ and $\hat{\sigma}^2_t$ as the sample mean and sample variance of the corresponding components of samples that exceed $\hat{\gamma}_t$. The mean vector $\hat{\mu}_t$ typically converges to the global optimum which is the vector of ones. The vector of standard deviations $\hat{\sigma}_t$ converges to the zero vector as the sampling converges to the degenerate distribution about the optimal solution.

2 PARALLEL CE

A common problem when solving complex optimization problems is the prohibitively large computational time required. For certain optimization problems, such as large phylogenetic tree construction, this time is typically of the order of days or weeks or more. One approach to decrease this computation time is to use an algorithm that has a parallel or distributed implementation.

The CE method is well suited to implementation in a distributed or parallel fashion due to its inherent parallel nature, however to date there has been no reported implementation of the CE method in a parallel fashion. In this section we will discuss our approaches to parallel implementation of the CE method and give examples for the Max-Cut and the Rosenbrock optimization problems.

We choose to use the Message Passing Interface (MPI) (Pacheco 1997, Gropp, Lusk, and Skjellum 1997) communication library for our implementation of parallel CE. MPI has become the de facto industry standard for programming parallel systems. It is also platform independent and so optimization routines written with MPI can be transfered to different architectures with relative ease.

The CE method performs two tasks: the generation of samples from the sampling density $f(\cdot;\hat{\mathbf{v}}_{t-1})$; and the updating of this density based on the samples. The first of these tasks can be parallelized in the following way. Suppose we wish to generate a sample of size N and are given s processors. Then each processor can generate a sample (from the same sampling distribution) of size $\frac{N}{s}$.

The second task of updating of the sampling density can be further broken down into two tasks: the scoring of the sample to identify the elite sample, and the updating of the sampling density based on the elite sample. To parallelize the scoring each CPU scores the sample which it has generated. Updating of the sampling distribution in parallel is not possible via the current algorithm 2.1. In order to complete this step we use a single CPU to update the sampling density and redistribute it to the other CPUs. In an effort to minimize the data transmitted between CPUs, the number of observations sent from each CPU is either the size of the sample generated by the CPU or the desirable size of the elite sample, whichever is lowest. For example, if $N = 1000$ and $\rho = 0.1$ we would want an elite sample size of 100. If we carried this out on two CPUs, each CPU would generate 500 observations and transmit its top 100 observations to a single CPU for the updating of the sampling density. However, if we used 20 CPUs each CPU would generate 50 observations and transmit all 50 to a single CPU for updating the sampling density.

Algorithm 2.1 (CE Algorithm for Parallel Optimization)**.**

1. *Choose initial parameter vector* $\hat{\mathbf{v}}_0$. *Set* $t = 1$.
2. *Generate on each of s CPUs a sample* $\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_{\frac{N}{s}}$ *from the density* $f(\cdot;\hat{\mathbf{v}}_{t-1})$.
3. *Pool the best* $\mathbf{min}(\frac{N}{s}, \rho N)$ *samples from each CPU to a single CPU and compute the sample* $(1-\rho)$*-quantile* $\hat{\gamma}_t$ *of the performance according to* $\hat{\gamma}_t = S_{(\lceil (1-\rho)N \rceil)}$.
4. *Using the pooled sample* $\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_N$ *solve the stochastic program (2) and denote the solution* $\hat{\mathbf{v}}_t$.
5. *If for some* $t \geq d$, *say* $d = 5$,

$$\hat{\gamma}_t = \hat{\gamma}_{t-1} = \cdots = \hat{\gamma}_{t-d}, \tag{6}$$

then stop; otherwise set $t = t+1$ *and iterate from Step 2.*

A modification to this algorithm is suggested in Section 4.1 which may allow for the partial parallelization of the updating of the sampling distribution.

2.1 MRIP vs SRIP

We consider two approaches to parallel optimization using the CE method. The first method, as described in Algorithm 2.1, splits the same optimization over multiple processors, and is called single replication in parallel (SRIP). The second approach is to run statistically independent replications of the same optimization algorithm, each on a different processor. This approach is multiple replication in parallel (MRIP).

The key advantages of the SRIP approach are as follows.

- The total elapsed time (the time elapsed between the start and the end of the program) for any single run is decreased, allowing results to be obtained faster.
- For large problems that could be above the memory constraints of a single processor, the division of the problem in an appropriate way will also divide the memory requirements over the processors, allowing for the optimization of larger problems.

In contrast, the key advantages of the MRIP approach are as follows.

- Communication requirements are much less than in the SRIP case, therefore when comparing the two methods with the same parameters and the same number of replications MRIP will take less time on average.
- Running the same optimization with a smaller sample size (for the MRIP) such that the total elapsed time taken for both the MRIP and SRIP approaches is the same has the potential to have the best of the n MRIP results being better than a single SRIP results.

Figure 1 shows the SRIP case for n processors. One processor communicates the initial sampling distribution parameter vector $\hat{\mathbf{v}}_0$ to all other processors. Each processor then generates a sample and evaluates the performance of each observation in the sample. The samples are communicated back to a single processor to update the parameter vector $\hat{\mathbf{v}}_t$ for the sampling distribution. This new parameter vector $\hat{\mathbf{v}}_t$ is then communicated to all other processors so

that they may draw a new sample. This process continues until the stopping criteria is satisfied.

Figure 2 shows the MRIP case for n processors. Each processor runs an independent simulation and produces statistically independent output.

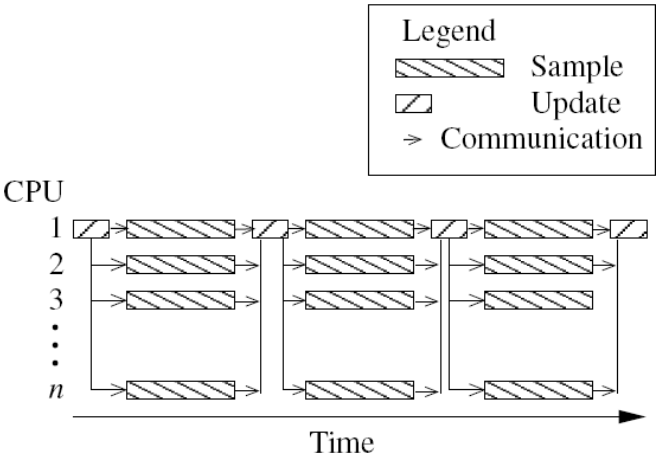

Figure 1: SRIP CPU timeline. A single processor initializes the sampling distribution and communicates this to all other processors. All processors then sample from this distribution before communicating their elite sample back to a single processor to update the sampling distribution.

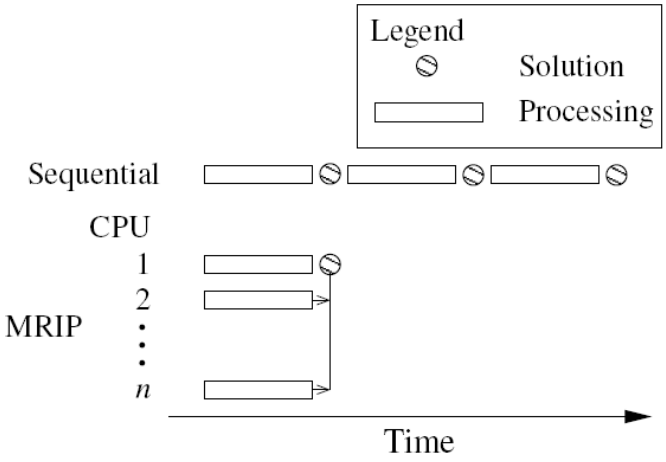

Figure 2: MRIP CPU timeline. All CPUs run a complete independent optimization before pooling their final results to a single CPU for reporting. The sequential line shows the longer total time taken to run each optimization one after the other.

3 RESULTS

Parallel speedup is defined as the ratio of the elapsed time for the sequential implementation to the elapsed time for the parallel implementation. A sequential implementation is one which runs on a single processor with no concurrent computations. Let S_p be the speedup factor for an algorithm run on p CPUs where the sequential version takes T_1 time, and the parallel version takes T_p time. Then,

$$S_p = \frac{T_1}{T_p}. \tag{7}$$

Due to variation in the running time for randomized optimization algorithms such as the CE method, in equation (7) the average running time over multiple runs is used. Table 1 shows the average running time from five runs of the parallel CE Max-Cut algorithm using various numbers of processors. Figure 3 shows the speedup factors for the Max-Cut combinatorial optimization problem with $n = 800$, $m = 400$ and an optimal solution of 160000. The speedup factor S_{16} is 13.2 which equates to 0.823 of linear speedup. The speedup factor S_{32} is 21.1 which equates to 0.658 of linear speedup. The proportion of linear speedup achieved starts to decrease after $p = 16$ processors. This can be seen in Figure 3 by the decrease in the gradient of the speedup

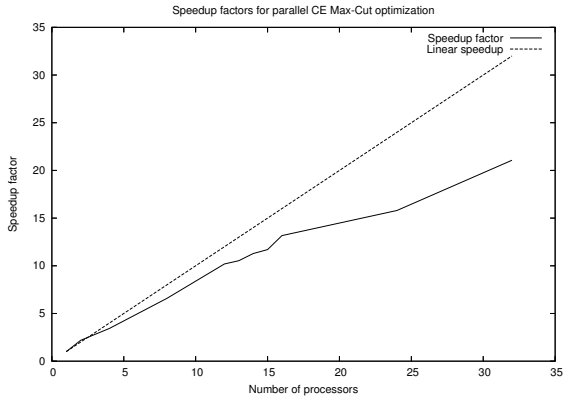

Figure 3: Speedup of the parallel Max-Cut CE algorithm. Run using an artificial network with $n = 800$ nodes and $m = 400$, the optimal score is 160000.

Table 1: Average running time of five individual parallel runs of the discrete Max-Cut CE algorithm on n processors along with the best and worst final scores.

Number of Processors	Average time (seconds)	Best score	Worst score
1	63.2	160000	160000
2	29.0	160000	159803
4	18.4	160000	160000
8	9.6	160000	160000
12	6.2	160000	160000
16	4.8	160000	160000
24	4.0	160000	159803
32	3.0	160000	160000

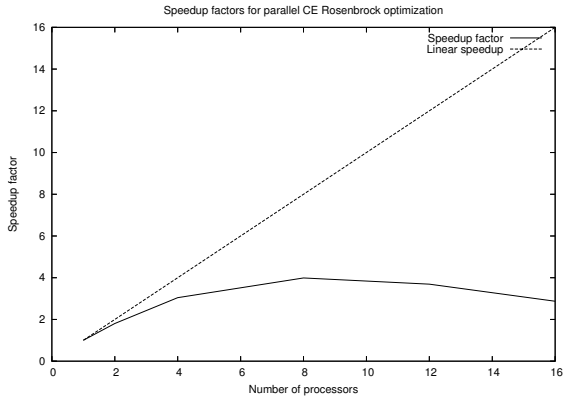

Figure 4: Speedup of the continuous 5-dimensional Rosenbrock CE algorithm on x processors.

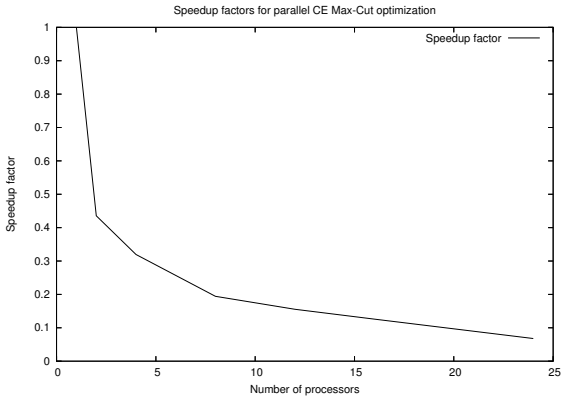

Figure 6: Speedup of the parallel Max-Cut CE algorithm. Run using an artificial network with $n = 100$ nodes and $m = 50$.

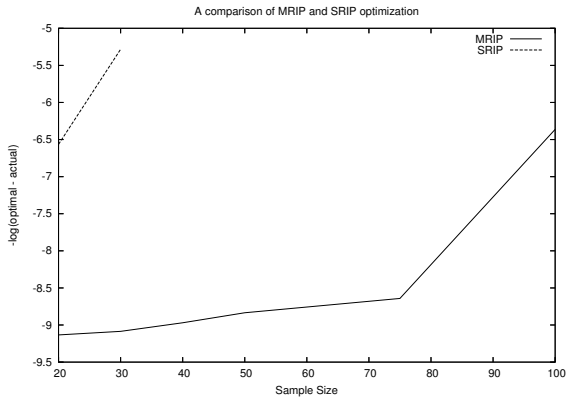

Figure 5: Comparison of the best scores for one SRIP run with sample size $10x$ and ten MRIP runs with sample size x of the Max-Cut CE algorithm on 10 CPUs.

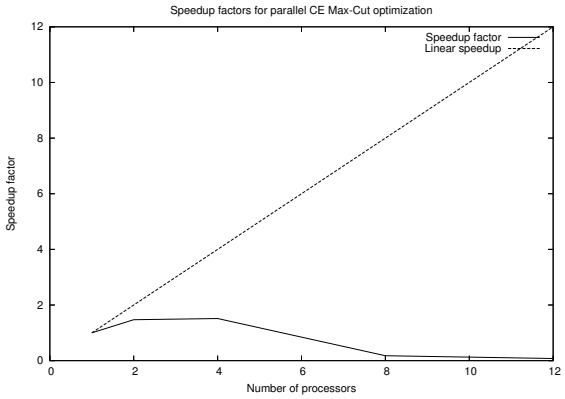

Figure 7: Speedup of the parallel Max-Cut CE algorithm. Run using an artificial network with $n = 400$ nodes and $m = 200$.

achieved line. A likely reason for this is that for a constant problem size the percentage of time for communication increases as the number of processors increases. Also, as the time taken to update the sampling distribution is constant for a fixed problem, the proportion of time spent each iteration performing this increases with the number of processors.

For a rough comparison, speedup factors of 11 to 19.4 were achieved for a network optimization problem using Parallel Move Simulated Annealing on 32 processors (Lee 1995). Directly comparing this with the parallel CE speedup of 21.1 for the Max-Cut problem shows the effectiveness of the parallel CE method. Although the problems optimized were not the same, they were both similar discrete optimization problems with the network optimization having greater complexity. The increased complexity should allow greater speedup factors due to more computation time being spent on the performance function evaluation which is parallelizeable. This is discussed further in Section 4.

Figure 4 shows the speedup factor for the parallel CE method applied to the 5-dimensional Rosenbrock problem. As can be seen from the figure, the speedup factors for the Rosenbrock problem are less than that of the 800 node Max-Cut problem when the number of processors exceeds 8. As the number of processors increases beyond 8 the speedup factors start to decrease slightly. This can be attributed to the size of the problem with our parameters when compared to the Max-Cut problem, being less as it is a lower dimensional problem.

The 'simplicity' of a problem directly relates to how well it can be parallelized. In general the simpler a problem, the lesser the proportion of the computational time spent generating and evaluating samples and the greater the proportion of time spent communicating and updating the

sampling distribution. This can be seen in figures 6, 7 and 3. All three are Max-Cut problems run on the same style artificial network, only the size of the network varies. Figure 6 shows a network of 100 nodes. The speedup factor decreases as the number of processors is increased for this size problem. Figure 7 shows a network of 400 nodes. While the speedup factor starts by increasing it quickly starts to decrease again once the number of processors exceeds 4. Figure 3 is the 800 node network discussed earlier. The speedup factors for this network increase all the way up to 32 nodes.

Given a particular optimization problem and a fixed amount of processor time, one may ask the question 'is it better to run the simulation multiple times and take the best result or run a single simulation for the same amount of processor time?'. To answer this we consider a comparison between MRIP and SRIP. For comparison purposes we will compare the best result from ten independent runs (MRIP) run in parallel, to a single parallel run with a sample size ten times larger (SRIP). Both these setups should take about the same amount of elapsed time on ten processors. Figure 5 shows the log of the difference between the simulation score and the optimal score against the simulation sample size for the Max-Cut problem. The first of the two lines in this figure reports results from single runs with ten times the sample size (SRIP), whereas the second line represents the best solution from ten independent runs (MRIP). The first line stops after a sample size of 30 because for larger sample sizes it produces the optimum solution and so the log of the distance is undefined. It can clearly be seen that in this example a single longer run produced the best results.

4 DISCUSSION

A key advantage of the CE method over other optimization methods is its ease of implementation for a diverse range of problems. This also applies to the parallel CE method.

When running a parallel CE implementation (SRIP) of the Max-Cut problem with 800 nodes on ten processors, a speedup factor of about 8 is achieved. This means ten independent runs, each on a single processor (MRIP), combine to use slightly less computation time than a single run with ten times the sample size running in parallel on ten processors. To make the computation times approximately the same a multiple of 8 times the sample size can be used for the SRIP implementation instead of ten. This still produces results on par with Figure 5.

There are several factors that can influence the speed-up factor for a particular problem. One of the key factors is the simplicity of the problem. When solving a very simple problem a greater proportion of the computation time is used in communicating the elite samples to a single processor and then updating the sampling distribution. Since updating the sampling distribution is not parallelizable and has to be carried out on a single processor, all other processors sit idle while this happens. As the problem gets more complex, the proportion of time taken to draw and score the samples increases, while the proportion of time taken to communicate between the processors and update the sampling distribution decreases.

Another factor is the computation time used to communicate. The main component of this is the sending of the elite samples from all processors to a single processor and the sending of the updated sampling distribution from the single processor to all other processors. As the number of processors increases, this communication overhead increases due to the larger total number of messages sent and received by the processors.

As more processors are used, the time for updating the sampling distribution remains relatively constant as this is performed on a single processor with a constant elite sample size. The time taken for each processor to generate a sample decreases as the number of processors increases due to the overall sample size being distributed over more processors. As mentioned above, the communication time increases as the number of processors increases. Thus the proportion of time each processor sits idle waiting for the new sampling distribution increases. Each processor will spend less time sampling, more time communicating and about the same amount of time waiting for a new sampling distribution, as the number of processors increases. Eventually, these factors will lead to a 'plateau' or even a decrease in the speedup factor as more processors are added.

4.1 Future Work

The next step is to parallelize the updating of the sampling distribution. This may be achieved through the calculation of a local update to the sampling distribution on each processor. Only these local updated sampling distributions are communicated to a single processor and not the whole elite sample. The single processor then treats these as partial sums to combine them to create the new global sampling distribution. For example, if our sampling distribution was a normal distribution, the new global mean μ_t would be the mean of the local means. The effect of this is that the global sampling distribution comes from a different elite sample when compared to pooling the observations before updating the sampling distribution. For example, observations not include in the calculation of one local sampling distribution may be 'better' than some observations included in another local sampling distribution.

The ability to have each processor generate a different sample size would be advantageous in a situation where processors may be of different speeds such as in a distributed environment.

We would like to implement more problems in a parallel CE fashion, specifically we would like to try biological problems such as large scale sequence alignment.

Lastly, we would like to develop a model that can predict the speedup factors expected when a problem is run on n processors.

5 CONCLUSION

We have shown that the CE method can be successfully implemented on a parallel computer using MPI and achieve good speed-up factors. This has been demonstrated with well known problems in both the continuous and discrete optimization cases. When comparing the speedup factors from a 800 node Max-Cut network using parallel CE to the speedup factors of a complex network optimization problem optimized with Parallel Move Simulated Annealing on 32 processors the parallel CE method performs better. The parallel CE method also has the added advantage of being an easy modification to any CE program.

A problems simplicity was a key factor in its ability to achieve reasonable speedup factors. The simpler a problem, the worse its parallel performance. This is not of critical importance as simple problems generally run in a short amount of time and do not require parallel implementation. As a problem becomes more complex, the computational time required increases as does the need for parallel implementation.

In a comparison of SRIP and MRIP for the example considered it is clearly better to run a single parallel simulation with a large sample size over multiple independent simulations with a smaller sample size. However, this may not be the case for problems that have a great degree of variability in their results. The Max-Cut example used consistently converges to the optimal solution and so does not display variability in its results.

ACKNOWLEDGMENTS

We wish to acknowledge the support of ARC Discovery grant DP0556631 and NHMRC grant Statistical methods and algorithms for analysis of high-throughput genetics and genomics platforms (389892).

REFERENCES

Gropp, W., E. Lusk, and A. Skjellum. 1997. *Using MPI: Portable parallel programming with the message-passing interface*. 4th ed. Cambridge, Massachusetts: The MIT Press.

Karp, R. M. 1972. *Complexity of computer computations*, Reducibility Among Combinatorial Problems, 85–103. Springer.

Lee, F. A. 1995. *Parallel simulated annealing on a message-passing multi-computer*. Ph.D. thesis, Department of Electrical Engineering, Utah State University, Logan, Utah.

Pacheco, P. S. 1997. *Parallel programming with MPI*. San Francisco: Morgan Kaufmann Publishers, Inc.

Rubinstein, R. Y., and D. P. Kroese. 2004. *The cross-entropy method: A unified approach to combinatorial optimization, Monte-Carlo simulation, and machine learning*. Springer.

AUTHOR BIOGRAPHIES

GARETH E. EVANS is a PhD student in the department of Mathematics at the University of Queensland, Australia. He completed his undergraduate degree in Mathematics and Computer Science at the University of Canterbury, New Zealand. His research interests include simulation and optimization of complex biological systems including but not limited to phylogenetics. His email address is <gevans@maths.uq.edu.au>.

JONATHAN M. KEITH is a post-doctoral researcher working with Prof. Kerrie Mengersen in the School of Mathematical Sciences at the Queensland University of Technology. He was awarded a PhD in Mineral Processing by the University of Queensland in 2000. He has been involved in various projects involving applications of Bayesian methods in bioinformatics and computational biology since 2000. His main current research interests are bioinformatic detection of non-coding RNAs in eukaryotic genomes and development of statistical methodology for high-throughput genetics and genomics platforms. He also maintains an active interest in stochastic optimization and its applications. His email and web addresses are <j.keith@qut.edu.au> and <http://www.uq.edu.au/∼uqjkeith/>.

DIRK P. KROESE has a wide range of publications in applied probability and simulation. He is a pioneer of the well-known *Cross-Entropy* method and coauthor (with R.Y. Rubinstein) of the first monograph on this method. He is associate editor of *Methodology and Computing in Applied Probability* and guest editor of *Annals of Operations Research*. He has held research and teaching positions at Princeton University and the University of Melbourne, and is currently working at the Department of Mathematics of the University of Queensland. His email and web addresses are <kroese@maths.uq.edu.au> and <http://www.maths.uq.edu.au/∼kroese/>.