GRIFFITH UNIVERSITY

# An Extended Extremal Optimisation Model for Parallel Architectures


## Author

Randall, M, Lewis, A

## Published

2006

## Conference Title

e-Science 2006 - Second IEEE International Conference on e-Science and Grid Computing

## DOI

10.1109/E-SCIENCE.2006.261198

## Downloaded from

http://hdl.handle.net/10072/13118

## Griffith Research Online

https://research-repository.griffith.edu.au

# An Extended Extremal Optimisation Model for Parallel Architectures

Marcus Randall
School of Information Technology
Bond University, QLD 4229
Australia
mrandall@bond.edu.au

Andrew Lewis
Institute for Integrated and Intelligent Systems
Griffith University
Australia
A.Lewis@griffith.edu.au

**Abstract:** A relatively new meta-heuristic, known as extremal optimisation (EO), is based on the evolutionary science notion that poorly performing genes of an individual are replaced by random mutation over time. In combinatorial optimisation, the genes correspond to solution components. Using a generalised model of a parallel architecture, the EO model can readily be extended to a number of individuals using evolutionary population dynamics and concepts of self-organising criticality. These solutions are treated in a manner consistent with the EO model. That is, poorly performing solutions can be replaced by random ones. The performance of standard EO and the new system shows that it is capable of finding near optimal solutions efficiently to most of the test problems.

**Keywords:** evolutionary and adaptive dynamics, extremal optimisation, parallel architectures

## 1 Introduction

Extremal optimisation (EO) is a relatively new metaphor for solving functional and combinatorial optimisation problems. It is based on the Bak-Sneppen model for self organising criticality (SOC) [3, 2]. To date only a limited number of combinatorial problems have been solved with this method. In this paper we propose an extended EO model that manipulates solution components as well as individual components. This is tested on the travelling salesman problem (TSP) as well as the multidimensional knapsack problem (MKP).

This paper is organised as follows. Sections 2 and 3 explain the EO algorithm and the concept of self-organising criticality. Section 4 describes how we extend EO so that it also operates at the population level (using self organising criticality) while Section 5 discusses the results of testing the new model across TSP and MKP instances. Finally, in Section 6 we conclude and discusses further research.

## 2 Extremal Optimisation

Extremal optimisation is one of a number of emerging biologically inspired metaphors for solving combinatorial optimisation problems. As it is relatively new in the field, compared to other techniques such as ant colony optimisation [9], genetic algorithms [11] and particle swarm optimisation [12], there exists wide scope to apply and to extend this meta-heuristic.

Boettcher and Percus [5, 6, 7] describe the general tenets of EO. We can view Nature as an optimising system in which the aim is to allow competitive species to populate environments. In the course of the evolutionary process, successful species will have the ability to adapt to changing conditions while the less successful will suffer and may become extinct. This notion extends to the genetic level as well. Poorly performing genes are replaced using random mutation. Over time, if the species does not become extinct, its overall fitness will increase.

For combinatorial optimisation problems, the genes represent solution components. In the case of TSP, solution components are the individual edges that make up a tour. In the original EO algorithm, the worst component would be replaced by one generated at random. However, as the performance of this version of the algorithm was unacceptable, $\tau$-EO was introduced [6]. In this form a poor component is chosen probabilistically rather than necessarily the poorest (at each generation). This is achieved using a parameter $\tau$ and Equation 1.

$$P_i = i^{-\tau} \qquad 1 \leq i \leq n \tag{1}$$

Where:

$i$ is the rank of the component,

$P_i$ is the probability ($P_i = [0, 1]$) that component $i$ is chosen and

$n$ is the number of components.

Ranking components from worst to best (i.e., rank one is given to the worst component), roulette wheel selection is used to choose the component to replace. For permutation problems such as the TSP, the way to replace the chosen node is not immediately obvious. Choosing a random node to swap this one with is likely to produce a worse result. To increase the chance of a better solution being produced, the worst edge connecting this node is first chosen and deleted. The node is then reconnected to its probabilistically chosen best neighbour. The third edge from this node (i.e., the one needed to be removed to ensure a valid permutation) is removed. The two remaining nodes that only have one edge each are then connected together. This procedure ensures a valid permutation.

### 2.1 Previous Applications

EO has been used more extensively in function optimisation than combinatorial optimisation. While Boettcher and Percus [5, 6] have described and carried out limited experimentation on the TSP, more successful application has been in graph (bi)partitioning [5] and the MAX-CUT (spin glass) problems [6]. For these at least, EO can locate optimal and near optimal solutions and is comparable to other meta-heuristics.

Beyond the original applications, some work has been done to adapt the standard EO algorithm to dynamic combinatorial optimisation. As an example, Moser and Hendtlass [16, 17] apply EO to a dynamic version of the composition problem. During the course of solving the problem with EO, it may undergo a variety of transformations to its structure and/or data. Despite the EO solver not being made aware of specific changes, it is able to adapt to them more readily than a standard ACS solver. This, however, was the reverse for the static version of the problem.

## 3 Self-Organising Criticality

The theory of self-organised criticality gives an insight into emergent complexity in nature. Bak [1] contends that the critical state is the most *efficient* that can actually be reached dynamically. In this state, a population in an apparent equilibrium evolves episodically in spurts, a phenomenon known as punctuated equilibrium. Local change may affect any other element in the system, and this delicate balance arises without any external, organising force.

EPSOC (Evolutionary Programming using Self Organised Criticality) [13] is a relatively new meta-heuristic that follows the method of Fogel [10] but with an additional selection operator from the Bak-Sneppen model. The steps of the algorithm are given in Algorithm 1.

Each set of values defining a trial solution is independent of all others. Therefore, the evaluation of trial solutions can be performed concurrently. Since the evaluation of the objective function generally dominates the execution time, Amdahl's Law ensures extremely high parallel efficiency.

**Algorithm 1** The EPSOC algorithm

1: Initialise a random, uniformly-distributed population, and evaluate each trial solution
2: **for** a preset number of iterations **do**
3: Sort the population by objective function value
4: Select a set, $B$, of the worst members of the population. For each member of $B$, add to the set its two nearest neighbours in solution space that are not already members of the set, or from the best half of the sorted population
5: Reinitialise the solutions of the selected set, $B$. For all other members of the population, apply some (generally small) random, uniformly distributed mutation.
6: Evaluate each new trial solution.
7: For each of the trial solutions, if it has a better objective function value than its original, retain it.
8: **end for**
9: **end**

EPSOC is a straightforward implementation of the nearest-neighbour, punctuated equilibrium model as an optimisation algorithm. By considering the trial solution vectors as defining a location in an $n$-dimensional space, the spatial behaviour of the model is realised naturally. The algorithm has a high degree of greediness as it maintains a large elite (in EPSOC, half the total population). This can be viewed as a constructive operator. It encourages gradual improvement in the better half of the population. Over time, the mutated population members move into the protected half of the population. Thus the median of the population moves toward better objective function values.

## 4 Extended Extremal Optimisation

The development of EPSOC allows us to investigate enhancing EO in an hierarchical hybrid of the two methods, an Extended Extremal Optimisation (EEO) algorithm. Experience with EPSOC has demonstrated that on a range of real-world problems in functional optimisation it has been able to provide solutions of better quality faster than representative algorithms from several other classes, provided it has sufficient parallel computing resources available. These qualities suggest it may be usefully applied as a convergence accelerant for EO, and may also improve the quality of solutions.

To construct the hybrid, a population of several, independent searches is implemented. This population is subject to EPSOC's evolutionary population dynamics and, at

each iteration, each member of the population carries out an EO-based search for some number of internal iterations to deliver an improved solution. The hybrid is effectively a *nested loop* of the two individual methods. Both EPSOC and EO use a user-specified, maximum iteration count for their termination criterion. The hybrid algorithm is thus simple, robust and easy to control. The conceptual operation of EEO is illustrated in Figure 1.

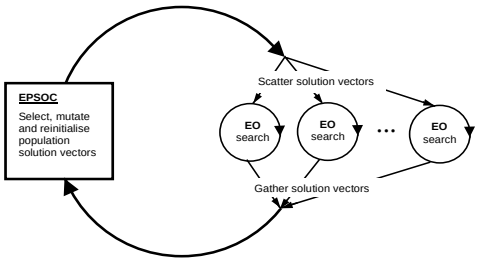


**Figure 1. Functional structure of EEO**

## 5 Computational Experience

Following Boettcher and Percus, we began our investigation of EO with an application to the TSP. A reasonably simple test case, a 26 city TSP fri26 drawn from TSPLIB [19], was used to evaluate our implementation. Early results were quite discouraging, as the algorithm displayed almost no ability to improve solutions at all. Figure 2 shows the objective function values obtained from 100,000 iterations of the EO algorithm. As can be seen, there is no displayed tendency toward improving cost (lower values in the figure) as the algorithm progresses.

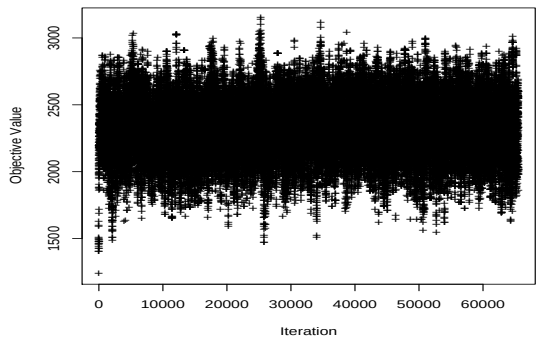


**Figure 2. Objective function values from EO on fri26 test case**

Figure 3 may give some insight into the reasons for EO's failure. It shows a histogram of sampled objective function values from the test. The global minimum for this test case, known to be a value of 937, lies more than 6 standard deviations from the mean of the sampled data, and the "tail" of the distribution can be seen to be vanishingly small.

While this cannot be considered a statistically significant sample - the total number of feasible tours is of $O(n!)$ - it demonstrates that the volume of the solution space that is *near-optimal* is an exceedingly small proportion of the total feasible solution space. Such a ratio will inevitably cause stochastic algorithms without some local search facility difficulty.

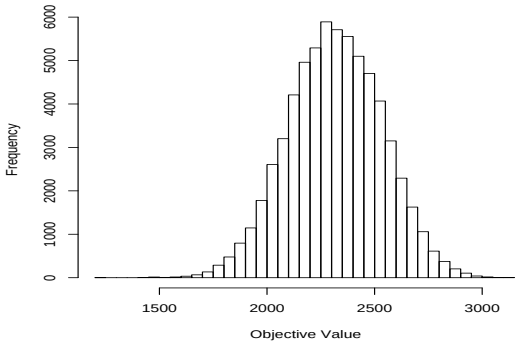


**Figure 3. Histogram of sampled objective function values for fri26 test case**

That EO performs poorly on TSP does not necessarily mean it is worthless, merely that it is poorly suited to this class of problems. It is a fundamental axiom of the No Free Lunch theorem [21] that all algorithms will perform poorly on *some* class of problems, and it has been demonstrated that problems can be constructed to favour any particular class of algorithms over another [20]. To further investigate the applicability of EO to combinatorial problems we tested our implementation on a multi-dimensional knapsack problem (MKP) [8].

MKP is an extension of the classical knapsack problem. In it, a mix of items must be chosen that satisfy a series of weighting constraints whilst maximising the collective utility of the items. Equations 2-4 show the 0-1 ILP model.

$$\text{Maximise} \sum_{i=1}^{N} P_i x_i \tag{2}$$

s.t.

$$\sum_{j=1}^{N} w_{ij} x_j \leq b_i \quad \forall i \quad 1 \leq i \leq M \tag{3}$$

$$x_i \in \{0,1\} \quad \forall i \quad 1 \leq i \leq N \tag{4}$$

Where: $x_i$ is 1 if item $i$ is included in the knapsack, 0 otherwise, $P_i$ is the profit of including item $i$ in the knapsack, $N$ is the total number of items, $w_{ij}$ is the weight of item $j$ in constraint $i$, $M$ is the number of constraints, and $b_i$ is the total allowable weight according to constraint $i$.

It might be asked what suggested EO would be any more successful on MKP than TSP. Sampling one of the test cases used in this study for 64,000 different random seeds produced the histogram of solutions shown in Figure 4.

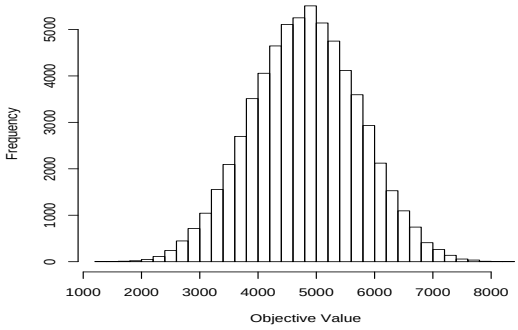


**Figure 4. Histogram of all sampled objective function values for mkp test case**

Comparing Figures 3 and 4, the shape of the distribution of solutions appears quite similar. But for MKP the optimal cost for the particular test case used was 6954 and from Figure 4 it can be noted that there are a significant number of solutions in the region of the global optimum, which lies only 2.3 standard deviations above the mean of the sample set, compared to more than 6 for TSP. Admittedly, not all of these are feasible solutions. From 256,000 random solutions only about 200 are feasible. Figure 5 shows the resultant histogram. While the graph is not very smooth, due to the small number of feasible samples, it can be noted that the global optimum is still only 2.6 standard deviations above the mean of the sample set. The number of solutions that lie in the *near*-optimal space can be expected to be larger than for TSP, allowing better performance for stochastic search methods.

The way which EO is applied to MKP is very much simpler than TSP. If the current solution is feasible, EO will select the probabilistic best item (in terms of cost). This may make the solution infeasible, in which case, EO will probabilistically select the lowest cost item to remove from the solution.

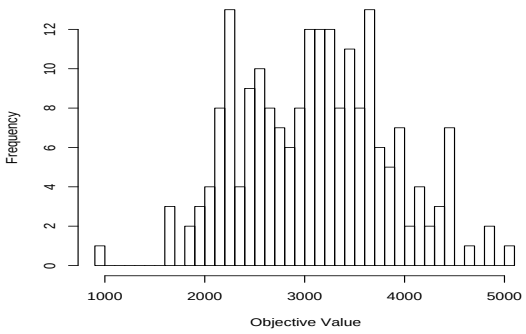


**Figure 5. Histogram of feasible sampled objective function values for mkp test case**

A number of MKP problem instances from OR-Library [4] were used as test cases. The problem instance descriptions are given in Table 1.

**Table 1. Problem instance parameters**

| Problem Name | N | M | Optimal Value |
|---|---|---|---|
| mknap1 | 6 | 10 | 3800 |
| mknap2 | 20 | 10 | 6120 |
| mknap3 | 28 | 10 | 12400 |
| mknap4 | 39 | 5 | 10618 |
| mknap5 | 50 | 5 | 6339 |
| mknap6 | 60 | 5 | 6954 |

These problem instances have been used in previous tests of Ant Colony Optimisation (ACO) algorithms [18]. In those tests, the ACO algorithm was able to find the global optimum in 4 of the 6 cases, only achieving results of 12380 on mknap3 (-0.2%) and 10137 on mknap4 (-4.5%).

EO was first tried by itself on the 6 test cases. The algorithm was allowed to run for 500,000 iterations, to ensure the best solution of which the algorithm was capable was returned. Best and median results obtained over a number of trials, and the average actual number of iterations required to obtain the results, are shown in Table 2 for each test case. EO has found the global minimum only for test case mknap1. This is, however, a trivial problem easily solved and thus not a particularly significant result.

At this point EO was combined with EPSOC to form the hybrid method, to investigate whether this could deliver better results in less time. The number of iterations to use at each level of the nested loop was determined by trial and error in the preliminary investigations reported in this paper, and there is perhaps potential for tuning this us-

**Table 2. EO-only test results**

| Problem name | Best result | Time taken | Mean result | Time taken |
|---|---|---|---|---|
| mknap1 | 3800 | 9140 | 3800 | 9140 |
| mknap2 | 6040 | 166304 | 5980 | 70808 |
| mknap3 | 12180 | 161089 | 12180 | 161089 |
| mknap4 | 9083 | 287120 | 9022 | 197651 |
| mknap5 | 6097 | 172585 | 6003 | 31215 |
| mknap6 | 6650 | 119523 | 6583 | 169988 |

ing self-adaptation [15]. However, observation of the convergence history of the hybrid algorithm tends to suggest that the gains from this may not be great. A typical convergence profile (for mknap6) for 100 iterations of EPSOC using 1000, 5000 and 50,000 iterations of EO is shown in Figure 6. The overall shape of each trace is determined by the EPSOC component and shows generally rapid initial convergence, followed by stagnation. Tests of EEO using 1000 iterations of the EPSOC component showed no benefit from allowing it larger numbers of iterations. The effect of changing the number of iterations for the inner, EO, component is primarily apparent in the lower starting values for the traces - by allowing EO to search further, even the first iteration of the hybrid algorithm is improved.

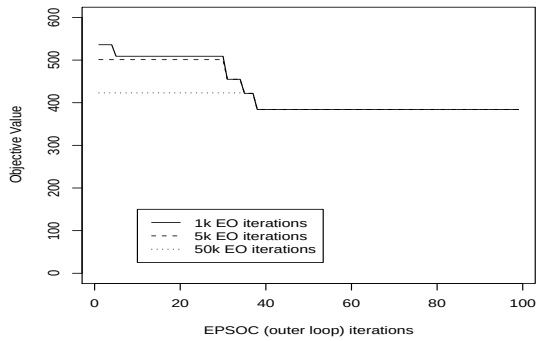


**Figure 6. Convergence histories of EEO for mknap6**

From Figure 6 it may be observed that the final result is obtained in less than 100 outer loop iterations for any of the number of inner loop iterations tested, i.e. using at most 100 iterations for the outer and at least 1000 for the inner loop would be adequate to achieve good results in minimal time. For this reason these limits were chosen for use for each of the test cases. The EPSOC component was given a population of 10 to manipulate, i.e., 10 simultaneous EO searches were executed on parallel computing resources at each iteration of the outer loop of the algorithm. The results achieved by the EEO hybrid algorithm are given in Table 3.

**Table 3. EEO test results**

| Problem name | Best result |
|---|---|
| mknap1 | 3800 |
| mknap2 | 6040 |
| mknap3 | 12210 |
| mknap4 | 10052 |
| mknap5 | 6107 |
| mknap6 | 6570 |

Both EO and EEO were able to find the global minimum for the trivial mknap1 test case. Comparison of Tables 2 and 3 shows that EEO achieved equal or better results than EO alone on all remaining tests cases except mknap6, for which the difference was only 1%. On all these test cases EEO also achieved these results in less time. The combined inner and outer loop iterations are equivalent in wall-clock time to 100,000 EO iterations. On average, EO required 80% more time to obtain generally poorer results.

Neither EO or EEO achieved the same success rate finding the global optimum as ACO. EEO results were, on average, within 1.1% for the simpler problems (mknap1-3) and within 4.9% for the harder problems (mknap4-6). The ACO algorithm was allowed 3000 iterations for each test case. However, because ACO constructs full solutions at each iteration, its computational requirements are substantially larger than EO or EEO. Table 4 compares indicative execution times, in CPU-seconds on identical hardware, for EO and ACO, for similar iteration counts on a few of the test cases. The execution times of ACO are proportional to $N$, as could be expected. The ratio between EO and ACO also appears roughly proportional to $N$, as EO does not construct a completely new solution but makes a single change to an existing solution, in essentially constant time.

**Table 4. Average execution times for EO and ACO**

| Problem name | EO | ACO | Ratio |
|---|---|---|---|
| mknap2 | 0.15 | 11.18 | 74.5 |
| mknap4 | 0.22 | 21.12 | 96.0 |
| mknap6 | 0.27 | 85.98 | 318.44 |

For the iteration counts for ACO and EEO noted above, the number of effective EO iterations will be 100,000 (1000 EO iterations in inner loop, 100 EPSOC iterations of outer loop), compared with 3000 ACO iterations, i.e., a factor

of about 33 times more. But from Table 4 it can be seen that the wall-clock execution time per iteration of ACO is consistently much greater than 33 times more than EO. For example, for mknap2 it is almost twice that and, as Table 4 shows, this ratio can be expected to improve as $N$ increases. As the execution time for EEO is governed by constant, user-selected iteration counts for the inner and outer loops, and the execution time of the inner, EO, loop its performance relative to ACO will improve similarly.

In summary, EEO has been found to deliver *near*-optimal results faster than an existing ACO algorithm, and the relative speed of the algorithm can be expected to improve for larger problem sizes. This class of algorithms, on the test cases considered, appears to quite rapidly approach solutions that are close to optimal, but then stagnate rather than converge to the global optimum. This behaviour has been observed using EPSOC for a bin-packing problem [14] and the suggestion has been made that the algorithms may well benefit from further hybridisation involving use of local search heuristics starting from the solution found by the evolutionary heuristics.

## 6 Conclusions

Extremal optimisation is an effective way of obtaining near optimal solutions to some combinatorial optimisation problems quickly. In this paper, we have proposed a population version of the algorithm that uses EPSOC to create and maintain EO solutions. Using the multidimensional knapsack problem we demonstrate that extended extremal optimisation presents advantages over its conventional counterpart. Further investigation will concentrate on the self-adaptation of parameters as well as hybridisation with local search engines.

## References


[1] P. Bak. *How Nature Works*. Springer-Verlag, New York, 1996.

[2] P. Bak and K. Sneppen. Punctuated equilibrium and criticality in a simple model of evolution. *Physical Review Letters*, 71:4083–4086, 1993.

[3] P. Bak, C. Tang, and K. Wiesenfeld. Self-organized criticality: an explanation of 1/f noise. *Physical Review Letters*, 59:381–384, 1987.

[4] J. Beasley. Or-library, 2005. Online at http://people.brunel.ac.uk/ mastjjb/jeb/info.html.

[5] S. Boettcher and A. Percus. Extremal optimization: Methods derived from Co-evolution. In *GECCO-99: Proceedings of the Evolutionary and Computation Conference*, 1999.

[6] S. Boettcher and A. Percus. Nature's way of optimizing. *Artificial Intelligence*, 119:275 – 286, 2000.

[7] S. Boettcher and A. Percus. Extremal Optimization: An evolutionary local search algorithm. In *Proceedings of the 8$^{\text{th}}$ INFORMS Computer Society Conference*, 2003.

[8] P. Chu and J. Beasley. A genetic algorithm for the multidimensional knapsack problem. *Journal of Heuristics*, 4:63 – 86, 1998.

[9] M. Dorigo and G. Di Caro. The ant colony optimization meta-heuristic. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 11–32. McGraw-Hill, London, 1999.

[10] L. Fogel. Autonomous automata. *Industrial Research*, 4:14 – 19, 1962.

[11] D. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison Wesley, Reading MA, 1989.

[12] J. Kennedy and R. Eberhart. The particle swarm: Social adaptation in social information-processing systems. In *New Ideas in Optimization*, pages 379–387. McGraw-Hill, London, 1999.

[13] A. Lewis, D. Abramson, and T. Peachey. An evolutionary programming algorithm for automatic engineering design. In *Proceedings of the Fifth International Conference on Parallel Processing and Applied Mathematics*, 2003.

[14] O. Mathieu. Utilisation d'algorithmes évolutionnaires dans le cadre des problèmes d'empaquetage d'objets. Master's thesis, FUNDP, Faculty of Sciences, Department of Mathematics, 2005.

[15] S. Meyer-Nieberg and H.-G. Beyer. Self-adaptation in evolutionary algorithms. In F. Lobo, C. Lima, and Z. Michalewicz, editors, *Parameter Setting in Evolutionary Algorithm*. Springer, 2006.

[16] I. Moser and T. Hendtlass. On the behaviour of extremal optimisation when solving problems with hidden dynamics. In *Proceedings of the 19th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, volume 4031 of *Lecture Notes in Artificial Intelligence*, 2006.

[17] I. Moser and T. Hendtlass. Solving problems with hidden dynamics - comparison of extremal optimisation and ant colony. In *IEEE World Congress on Computational Intelligence*, 2006.

[18] M. Randall. A dynamic optimisation approach for ant colony optimisation using the multidimensional knapsack problem. In *Recent Advances in Artificial Life*, volume 3 of *Advances in Natural Computation*, pages 215 – 226. World Scientific, 2005.

[19] G. Reinelt. TSPLIB - A traveling salesman problem library. *ORSA Journal on Computing*, 3:376–384, 1991.

[20] O. Sharpe. Beyond nfl:a few tentative steps. In J. Koza, editor, *Proceedings of the Third Annual Genetic Programming Conference*, 1998.

[21] D. Wolpert and W. Macready. No free lunch theorems for search. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.