Extremal Optimisation for Assignment Type Problems

Marcus Randall[1], Tim Hendtlass[2], and Andrew Lewis[3]

[1] School of Information Technology
Bond University
Queensland
Australia
mrandall@bond.edu.au

[2] Faculty of Information and Communication Technology
Swinburne University of Technology
Victoria
Australia
thendtlass@swin.edu.au

[3] Centre for Integrated and Intelligent Systems
Griffith University
Queensland
Australia
a.lewis@griffith.edu.au

Summary. Extremal optimisation is an emerging nature inspired meta-heuristic search technique that allows a poorly performing solution component to be removed at each iteration of the algorithm and replaced by a random value. This creates opportunity for assignment type problems as it enables a component to be moved to a more appropriate group. This may then drive the system towards an optimal solution. In this chapter, the general capabilities of extremal optimisation, in terms of assignment type problems, are explored. In particular, we provide an analysis of the moves selected by extremal optimisation and show that it does not suffer from premature convergence. Following this we develop a model of extremal optimisation that includes techniques to a) process constraints by allowing the search to move between feasible and infeasible space, b) provide a generic partial feasibility restoration heuristic to drive the solution towards feasible space, and c) develop a population model of the meta-heuristic that adaptively removes and introduces new members in accordance with the principles of self-organised criticality. A range of computational experiments on prototypical assignment problems, namely generalised assignment, bin packing, and capacitated hub location, indicate that extremal optimisation can form the foundation for a powerful and competitive meta-heuristic for this class of problems.

1 Introduction

Extremal Optimisation (EO) [6, 8] is a meta-heuristic search technique that has its origins in the science of self-organised criticality (SOC). SOC has been used to describe behaviour in systems as diverse as geophysics, economics and biological evolution. It is only recently that these concepts have been applied to solving optimisation problems [6].

This chapter investigates and examines the use of EO for a class of problems known as the assignment type problems (ATPs). We use these problems as our benchmark problems as they often have difficult constraints. Initially we look in detail at how EO selects and performs moves in search space on the bin packing problem. We then further develop EO to show that with suitable support heuristics it is capable of producing solutions comparable to those of more established meta-heuristics.

The remainder of this chapter is organised as follows. Section 2 briefly explains the idea of SOC while Section 3 gives an overview of the general tenets of EO. Section 4 describes assignment type problems (ATPs). Given that there has been little work done on the analysis of EO's search behaviour in the literature, a detailed case study examination of EO transitions on one of the test problems, bin packing, is undertaken in Section 5. Section 6 shows how EO can be used more generally on ATPs. A number of topics are addressed that EO potentially needs to help it become competitive with more established meta-heuristics. These are, specifically, transition operators, partial feasibility restoration, and population models. Computational experiments across a range of problem types and instances are discussed in Section 7. Finally, the conclusions and future research directions are given in Section 8.

2 Self-organised Criticality

Self-organised critical behaviour can be observed in systems in which, over long periods of time, seemingly only small changes take place. However, these small changes gradually build to a critical state that can eventually trigger large events in a domino fashion. The most commonly related example of self-organised criticality is the sand pile model [2]. Adding a grain of sand at a time to the pile builds it up slowly to a point where the addition of another grain will push its downhill neighbour grains, which in turn pushes other grains – in effect triggering a sand slide or *avalanche*. The pattern of this behaviour is present in a diverse range of natural and artificial systems, including the flow of rivers, and the formation of mountain landscapes and coastlines and economic fluctuations in stock markets [4].

There are two important characteristics of self-organised critical systems. The first is that they do not require fine tuning of parameters to exhibit complex, self-organised behaviour. The second is that the avalanche magnitude divided by the log of the time between avalanches of this size is roughly constant and follows a pattern whereby larger events are exponentially less likely than smaller events. A good illustration of this is the Guttenberg-Richter Law for the size of earthquakes [2], an earthquake of size 4 on the (logarithmic) Richter scale will occur ten times more frequently than an earthquake of size 5.

Our interests in this chapter are the applications of SOC within biological evolutionary systems that can be ultimately simulated in order to provide models for

combinatorial optimisation. Bak [2] has noted that the concept of survival of the fittest, or conversely, the selection and elimination of the few most poorly adapted species in a particular environment, displays the characteristics of SOC. In other words, there is no central organising agent nor finely tuned system in Nature that manages the survival or extinction of the species. The latter is referred to as *adverse selection* [7] and is a property that can be modelled in terms of SOC. The Bak-Sneppen model [3] represents a simplified system of interacting species. The underlying purpose of the model is to demonstrate emergent behaviour in evolutionary selection processes. One key simplification is that species are represented by a single fitness value, which is not derived from a genetic representation of the species. All species are assigned a fitness value in the range $[0, 1]$, where 0 indicates the least fit. At each step of the evolutionary process, the species with the lowest fitness value has this fitness changed to a random value. As in this simple model there is no structure to a species, this is the biological equivalent to allowing the original species to become extinct and a new species to take its place.

The model also recognises that species do not exist in isolation. For example, if a species becomes extinct, the species directly above and below it in the food chain will be affected. Therefore the neighbours (as defined by a set of lattice or ring sites) of a species that has changed will have their fitness values updated to random values as well. After many steps of this process, all species will have had their fitnesses increased and the probability increases that the species that have their fitnesses replaced have them replaced with lower values. Again it is probable that the new worst fitness is one of those just introduced and that the replacement process will result in a sharp reduction in a neighbour's fitness. Within a few steps like this the average fitness of the species collapses, and then the process of the gradual increase of fitness values begins again. This sequence of events is referred to as a *punctuated equilibrium* [3]. That is, apparent equilibrium in the system is punctuated by avalanches. Such events allow the species to potentially sample all of configuration space.

This process can be shown with a remarkably simple computer simulation. Algorithm 1 shows Bak-Sneppen's [4] ring model[4] in which each species affects its two neighbour species. Neighbours are defined in terms of position on the ring. The values of the worst species on the ring (and its neighbours) are replaced by other random values each iteration, and the worst fitness of all of the current species is reported together with the best worst species fitness found in any iteration so far. Figure 1 shows a typical run of the algorithm. The connected line "envelope" function represents the highest value of the worst species value found up to that particular iteration. The jumps from a previous maximum to the next mark the occurrence of an avalanche. The reason for this can be deduced from consideration of the distribution of values within the population. For the Bak-Sneppen model, the population has a uniform random distribution of fitness values between a current, lower bound, $\lambda_{min}(t)$ and 1, as might be represented by values within the solid outline in the histogram in Figure 2. For the lower bound to "jump" from its present value, $\lambda_{min}(t)$ to the value at the upper end of the filled region in the figure, $\lambda_{min}(t+1)$, *all* the species that currently lie in the filled region must migrate above that bound (and the histogram will expand upward to fill the shaded region in Fig-

[4] The neighbour of the last species is the first species, and vice versa.

ure 2.) This evidently cannot be achieved in a single iteration; several iterations will be required as successive species "cascade" to higher fitness levels.

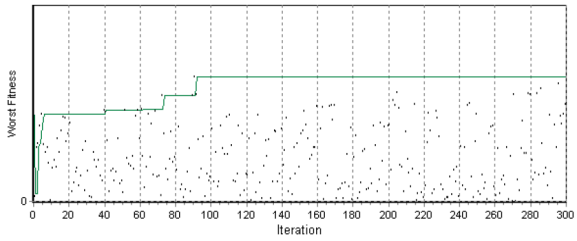

Fig. 1. The output of a typical run of the SOC algorithm. The connected "line" is the envelope function. Each point in the graph represents the worst value at that particular step.

Algorithm 1 The SOC algorithm.

Generate a random vector *species* in the range $[0, 1]$
for each generation **do**
  Find the species with the worst/lowest value
  Find the neighbour directly below this worst species
  Find the neighbour directly above this worst species
  Generate new random values for these three species
  Report the worst value of any species
  If this worst value is higher than any worst value found in previous iterations, update the best worst value found so far. Report the current best worst value.
end for
end

To apply the concepts of SOC to optimisation problems, it is necessary to define a mapping between a fitness value and the structural components of a species. Essentially, this puts back the details that Bak and Sneppen discarded in their model. One such example is extremal optimisation and is discussed in detail next.

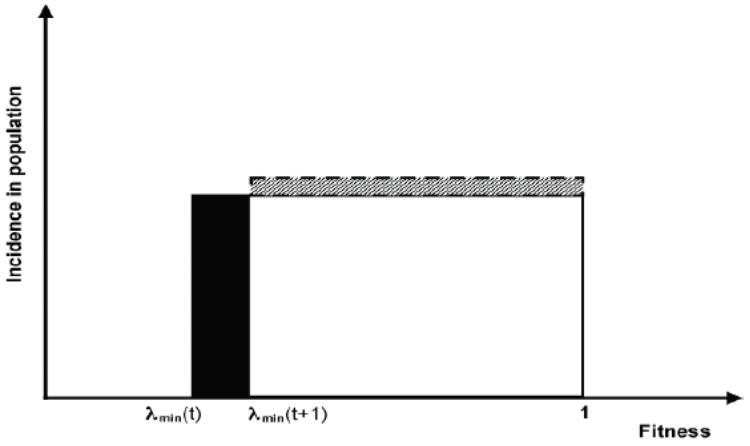

Fig. 2. Distribution of fitness values for a sample population of species in the Bak-Sneppen model.

3 Extremal Optimisation

Extremal optimisation is one of a number of emerging Nature inspired metaphors for solving combinatorial and continuous optimisation problems. As it is relatively new and unexplored, compared to other techniques such as ant colony optimisation (ACO) [13], genetic algorithms (GAs) [18] and particle swarm optimisation (PSO) [22], there exists wide scope to test and to extend its capabilities. Unlike its counterparts, the canonical algorithm manipulates a single solution rather than a population of solutions. Additionally, it never converges as the single solution is continually changing (see Section 5 for a demonstration of this).

EO is loosely based on the principles of the Bak-Sneppen model and simulates the notion that some species flourish while others do not [6, 8, 10]. This form of selection is also present at the genetic level. We can use a mapping between 'genes' (or species structural components) and 'solution components' to describe the general operation of EO to combinatorial optimisation problems. Solution components are the building blocks of the solution, some examples being an agent assigned to a particular job for generalised assignment, or the inclusion of an item in a knapsack for the knapsack problem. In the original version of the EO algorithm, at each iteration, the component whose fitness is worst, would be replaced by another solution component generated at random. In essence, however, this choice of always selecting the worst component to modify leads to too greedy a search, and consequently its performance was poor. Like other meta-heuristic algorithms, an element of randomness (in the form of probabilistic selection) was introduced. This became known as $\tau-$EO. Components are ranked from worst (rank 1) to best (rank n). The parameter τ and the rank controls the selection probability for each solution component [8]. This is achieved using Equation 1. It is evident that lower ranks will receive larger values than higher ranks.

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

Where:

i is the rank of the component,
P_i is the probability ($P_i = [0, 1]$ when normalised) that component i is chosen and
n is the number of components.

Values of τ close to or equal to zero produce a random search strategy. Conversely, allowing $\tau = \infty$ gives the original EO algorithm. Algorithm 2 shows the mechanics of a single $\tau-$EO iteration.

Algorithm 2 A single $\tau-$EO iteration. Note that vector P need only be calculated once according to Equation 1.

Rank the solution components from worst to best
j = Select a ranked component using roulette wheel selection on normalised P
Assign x_j a random (legal and different) value
end

This procedure is performed a fixed number of times or until a particular solution quality is reached.

3.1 Existing EO Applications

Compared to other recent meta-heuristics, particularly ACO and PSO, extremal optimisation has received relatively little attention. Below is a representative summary of existing applications of EO.

Boettcher and Percus [6, 8] have described and carried out limited experimentation on the travelling salesman problem (TSP). However, more successful application has been in graph (bi)partitioning [6] and the max-cut (spin glass) problems [8]. For these at least, EO can locate optimal and near optimal solutions for the investigated test cases and is comparable to other meta-heuristics.

Randall and Lewis [31] present an extended form of EO in which it is used as a sub-ordinate heuristic by another meta-heuristic known as Evolutionary Population Dynamics (EPD). Experiments on small multi-dimensional knapsack problem instances showed that EO with EPD achieved equal or better results than EO on almost all tests cases. The results using EO with EPD were also compared to tests using a standard ant colony optimisation solver. EO with EPD was been found to deliver near-optimal results faster than the existing ant colony algorithm.

Randall [29] has performed an initial investigation of EO for the generalised assignment problem (GAP). In that paper a simple population model for EO was presented along with a heuristic that altered solutions so as to reduce the amount of their infeasibility (a process known as *partial feasibility restoration*). This heuristic helped EO to produce very good quality solutions. In fact, both the canonical EO and population versions were able to find statistically significantly better solutions than a state-of-the-art ant colony system (ACS) implementation, a very efficient meta-heuristic for this problem.

A variation of the bi-partitioning problem used for community detection is solved using EO by Duch and Arenas [14]. Specifically, they use it to optimise the modularity and to identify the communities of complex networks. However, as reported in Xiaodong, Cunrui, Xiandong and Yanping [33], this implementation is sensitive to the initial solution, prone to being trapped in local optima, and not yet competitive with particle swarm optimisation.

Middleton [25] proposes a modification to standard EO that makes it better capable of solving the Ising spin glass problem. This is referred to as *jaded EO* and increases the fitness of a spin in proportion to the number of times it has been flipped. Empirical results, in comparison to standard EO, show that it is significantly better.

Beyond the applications to benchmark problems, some work has been done to adapt the standard EO algorithm to dynamic combinatorial optimisation. As an example, Moser and Hendtlass [26, 27] 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.

Moser and Hendtlass [28] propose an EO implementation for dynamic aircraft landing. This problem consists of two related parts, namely determining the order/permutation of planes to land on a single runway, and assigning landing timeslots for the planes as they enter the horizon of air traffic control (but taking other planes into account). The latter is a deterministic problem, so EO is used for the

former. A set of K candidate solutions is generated, where K is the number of aircraft on the horizon. Each of these is generated by swapping the landing orders of two aircraft. These candidates are ranked according to how close the landing times resemble the planes' target times. A new solution is chosen according to EO's rules. The results showed that this EO implementation could outperform a range of meta-heuristic applications.

Galski, de Sousa, Ramos and Muraoka [17] present an EO algorithm to find the optimal design of a simplified configuration of a thermal control system for a spacecraft platform. The objectives are to minimise the difference between target and actual temperatures on radiation panels as well as to minimise battery heater power dissipation. Using EO, two possible solutions were found. As the authors combined both objectives into a single function, the designs were really only able to satisfy the first objective. Future work will optimise both objectives using a Pareto front approach.

Another novel application of the meta-heuristic is to the protein folding problem. Shmygelska [32] implements a two stage process in which EO finds a good starting solution, in terms of pairwise arrangements of amino acids, from which a local Monte Carlo based search can find a refined solution. The results compare favourably with a known random search technique specific to this problem.

EO has also been adapted to solve continuous optimisation problems by Zhou, Bai, Cheng and Wang [34]. This implementation concentrates on the Lennard-Jones clustering problem. EO is used as a global optimiser, selecting probabilistic worst components (or "atoms") to change and then using a gradient based local search to improve the solution. The difficulty is that their approach does not scale well. Nevertheless, it is competitive with other heuristic methods for smaller problems.

It is clear from the above survey that to date EO has really only been applied to problems that are relatively unconstrained. The mechanics of the algorithm, particularly those concerned with giving a solution component a new random value, make it difficult for EO to naturally process more constrained problems. One class of such problems, the assignment type problems (discussed next), contain a range of interesting real-world constraints (such as capacity and group related constraints). In light of this, the extensions of the EO paradigm proposed in this chapter will allow it to be more widely and generally applicable.

4 Assignment Type Problems

"Assignment Type Problems" (ATPs) [12] are a collection of optimisation problems for which a number of items are to be assigned to groups subject to a set of resource/capacity constraints. They include the generalised assignment, bin packing and capacitated hub location problems to name a few. These three problems are broadly representative of ATPs and will form the test problems for the experimental work within this chapter. Brief descriptions of each follow:

- *Generalised Assignment Problem* – The generalised assignment problem [24] is a problem in which jobs are assigned to agents for these agents to perform subject to capacity constraints. Each job may be performed by one agent only. The aim is to minimise the total cost of assigning the jobs to the set of agents.

- *Bin Packing Problem (BPP)* – The bin packing problem has different variations. The one considered here is the standard one-dimensional version [21]. A set of items, each of which has a particular weight, is packed into a number of bins. Each bin has the same weight capacity. Two objective functions are possible, both of which are used in this chapter. The first is used in Section 5 and treats bin packing as a constraint satisfaction problem in which the total amount of excess weight for a fixed number of bins, is minimised. The second is used in the remainder of the chapter and explicitly minimises the number of bins.
- *Capacitated Single Allocation Hub Location Problem (CSAHLP)* – The CSAHLP belongs to a general class of hub and spoke problems for which the aim is to efficiently transfer large quantities of commodities (such as passengers, mail and telecommunication traffic) between each node pair of a network. A subset of the nodes (called *hubs*) act as consolidation centers for bulk transfers. The CSAHLP is a difficult variant of this problem in which the number of hubs is not fixed, and each hub has a limited flow capacity. The mathematical formulation of it may be found in Equations 6 – 14 of Randall [30]. Furthermore, this paper describes a number of generic support heuristics (such as node-to-hub allocation and feasibility restoration). These will be used in our enhanced version of EO.

5 A Detailed Examination of EO on Bin Packing

As previously mentioned, EO is a relatively new heuristic under-explored compared to many others. Despite using a seemingly simple move mechanism, the following step-by-step analysis reveals that it does not always bias towards the most favourable solutions. An archetypal example of an assignment problem, bin packing, is used to demonstrate this behaviour. This is solved as a constraint satisfaction problem, in which the amount of excess weight in the bins is to be minimised, with zero being considered optimal.

EO changes its solution in a series of moves with the single solution moving through problem space. At any time there are a series of legal moves available to it. The move that is made is one that pushes away from the current position by randomly altering a poor component of the current solution. The direction in which the push takes the solution is random, but by always altering bad component values. The long term trend is towards better solutions (ones with lower constraint violation) but in a very non-monotonic way.

In terms of bin packing, EO will select one overfull bin and randomly take one item from that bin and transfer it to another randomly chosen bin. If the total amount of excess weight in all the overfilled bins is thus reduced this might be referred to as a good move, if the total amount of excess weight is increased this is a bad move (the occasional moves in which the total of excess weight does not change may be referred to as a neutral move). For the problem u120_00 [5, 16], checking all possible moves that could be made at each point of a thousand step search[5]

[5] All the figures in this section are based on the results from one hundred independent repeats, each consisting of one thousand actual moves (sufficient to allow stagnation to occur if it is going to occur). At each actual move either all possible legal moves or one thousand random test moves are trialled but then reversed after the data have been collected for the statistics.

through the problem space shows the ratio of good to bad moves to be 0.51 to 1. If on the other hand the bin from which an item is to be removed may be chosen probabilistically from the N available bins (ranked from worst to best), then the probability of choosing bin n is

$$P_n = \left(R\left(\frac{w_n}{\sum_{i=1}^{N} w_i} \right) \right)^{-\tau}$$

where w_i is the weight of bin i, R is the ranking function and τ, the only user specified parameter for EO, determines how much the algorithm concentrates on the most overfilled bins. If τ is set to 1 then overfull bins are treated equally. If τ is set to 1.4 as recommended in Boettcher and Percus [9] the good to bad ratio is changed to 1.17 to 1.

Knowing the ratio of good to bad moves is only part of the picture, one also needs to know how good and how bad these moves might be. Figure 3 shows histograms of both the good and bad moves for EO. Note that the distributions of the two types of moves are very different. The bad moves are worse (on average) than the good moves are good.

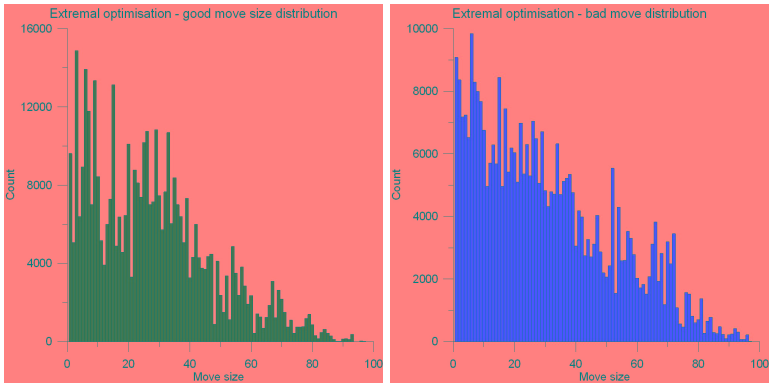

Fig. 3. Histograms of good (left) and bad (right) moves available to EO.

For comparison, consider the choices available to two other algorithms, random hill climbing and a genetic algorithm on this same problem.

Random hill climbing also uses a single solution and creates an endless series of random candidate solutions, and, with a certain probability, a candidate solution replaces the current solution if it is better (i.e., has fewer constraint violations). The replacement probability should be between 0.5 and 1, the higher this figure the greedier the algorithm. For u120_00 and a probability of 1, the ratio of good possible moves to bad possible moves is 0.08 to 1. Figure 4 shows the histograms of the good and bad moves for a replacement probability of 1 (i.e., always replace if better). The low number of good moves (cf. the number of bad moves) means that few changes

are kept and that the time taken to find the relatively small sequence of good moves necessary to climb a local optimum may be high.

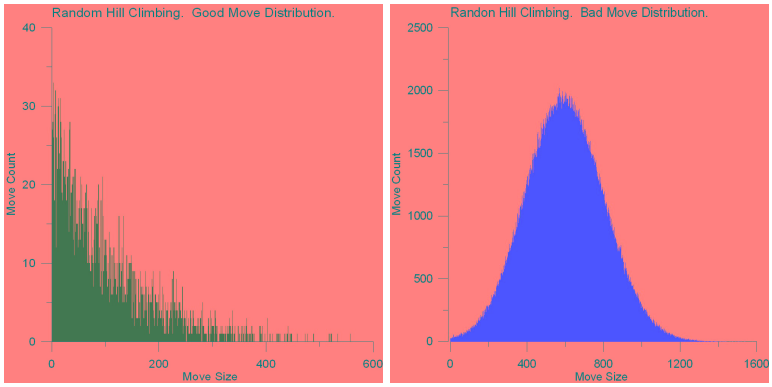

Fig. 4. Histograms of good (left) and bad (right) moves available to random hill climber.

A genetic algorithm uses a population of solutions and builds new solutions by combining parts of existing solutions together with some mutation. For the results quoted here, two parents are used and selected by a simple tournament between two candidates, with a probability of 0.7 of choosing the fittest candidate as a parent. Using single point crossover between the two parents with a probability of 0.1 of randomly moving one item in the solution to another bin (mutation), the ratio of good to bad moves for u120_00 is 5.7 to 1. Figure 5 shows the histograms of these good and bad moves. The high ratio of good to bad moves means that a GA will have the least trouble of the three algorithms considered in putting together the sequence of good moves to find a good quality solution.

Figures 6, 7 and 8 show the relative proportion of good, bad and neutral moves available at each of the thousand steps that make up a single run of each algorithm. Figure 6 shows that the relative prevalence of these moves for EO does not vary markedly during the run, while Figure 7 shows that the random hill climber initially has many good moves available but rapidly stagnates in a local suboptimal position. Initially the GA has many good moves available but then stagnates. There follows a cyclic series of steps. At each step an increasing number of good moves become available as more of the population climb a (probably) suboptimal peak. There is then a period of stagnation ending with a sudden breakthrough as progress is made towards a better (but probably still sub-optimal) peak after which the cycle repeats. With each breakthrough the fitness of the local sub-optimal peak improves so that the probability of another breakthrough decreases (i.e., the time between breakthroughs increases).

Considering that it will take a number of good moves in succession – a number that depends on how large each of the good moves is – it is clear that the probability of this occurring is far higher for EO (with the good to bad ratio of 1.17:1) than

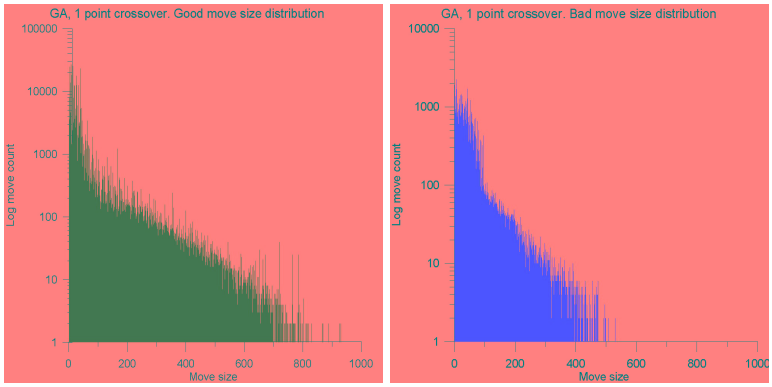

Fig. 5. Histograms of good (left) and bad (right) moves available to a genetic algorithm.

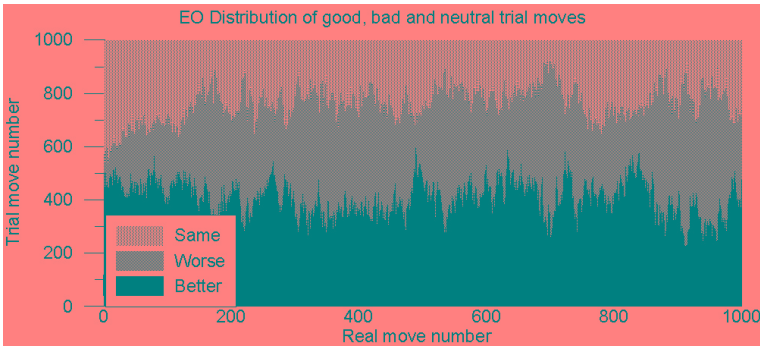

Fig. 6. Relative availability of good, bad and neutral moves during a run of EO.

with random hill climbing (0.08:1) except in the early stages of the search. A genetic algorithm has more good moves available with an overall average of 5.2:1, a ratio that is far higher for some short periods. What is in favour of using EO is the fact that, unlike the other two (greedier) algorithms, it does not reach one (or cycle round a series of) local sub-optimal position(s) and then stagnate (forever in the case of the random hill climber or until a fortuitous breakthrough occurs for the genetic algorithm). EO's single solution never stagnates but continually moves in problem space as is evidenced by the fact that the good to bad ratio is consistent during the run.

Figure 9 shows that EO very rarely revisits previously explored positions, unlike the other two algorithms. The low probability of EO constructing a series of good moves means that EO should not be expected to work well if run alone. The fact

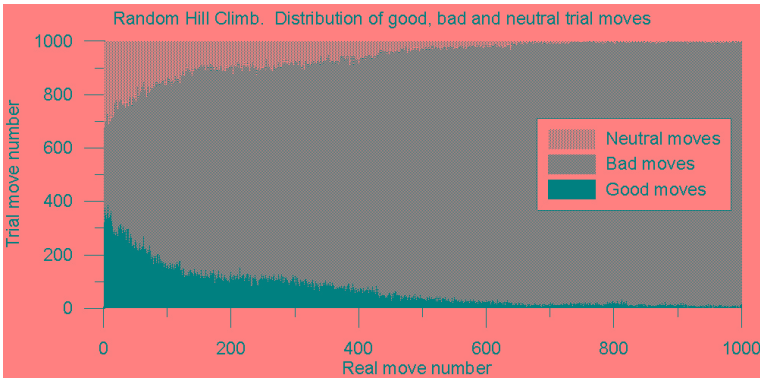

Fig. 7. Relative availability of good, bad and neutral moves during a run of random hill climber.

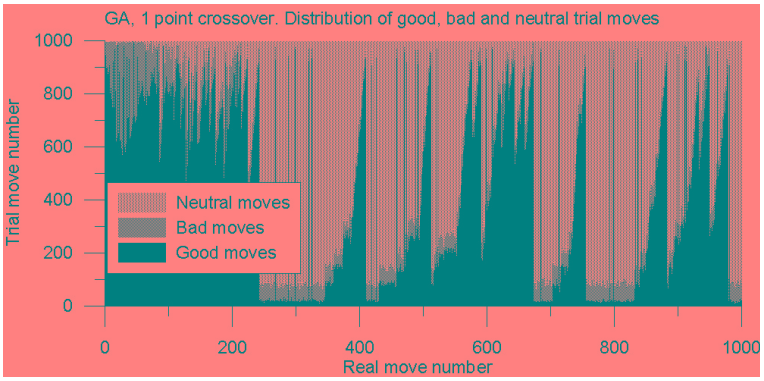

Fig. 8. Relative availability of good, bad and neutral moves during a run of a genetic algorithm.

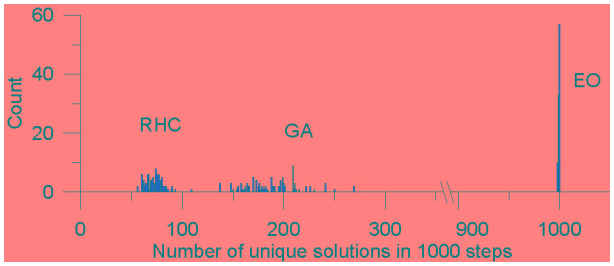

Fig. 9. The number unique places visited by each algorithm during 1000 steps. The histograms are built from the results of 1000 independent repeated runs.

that it does not stagnate suggests that it would be a good component of a meta algorithm that:

- contains another element that can lift the good to bad ratio so that EO will be driven further up local peaks (as long as this does not also inhibit EO's ability to migrate endlessly through problem space), and
- runs other, more local, search algorithms in collaboration with EO, primarily relying on EO to find regions of interest for these other techniques to explore more fully.

An endless series of random solutions would show a very low probability of revisiting the same place in problem space in any reasonable length run and one must ask what advantage there is in using EO over a series of fully random solutions. The answer lies in the average quality of the places in solution space visited by the two. Table 1 shows that there is a clear difference in the average quality of the start points each algorithm would provide for other, more local, search algorithms while Figure 10 shows how the fitness varies during a typical run. Although the ratio of the fitnesses is only a little over 2, if fitness was to be graphed against the number of solutions with each fitness value, an inverse bell shaped distribution would be produced. If the number of solutions with this or worse fitness is considered, extremal optimisation is ahead by orders of magnitude.

	Random Moves	Extremal Optimisation
Average fitness	1833.3	783.6

Table 1. Average fitness of a thousand moves. As this is a minimisation problem, the smaller value obtained by EO is better.

The discussion above has been written using data obtained from runs attempting to solve a bin packing problem. A similar series of runs, this time using a graph colouring problem, exhibit similar behaviour. This indicates that the observations made are predominantly a result of the behaviour of the algorithms used and not so much of the problems themselves. Thus the observations might be expected to substantially hold for a wide range of assignment problems.

6 Applying EO to ATPs

The previous section showed the performance of canonical EO on one of the test problems. EO was capable of exploring search space nearly without revisiting previous solutions. However, the results were not of an acceptable standard. Too many of the potential moves lead to poor quality solutions. Consequently, the level of constraint violation (for that particular version of bin packing) was such that feasible solutions could not be easily produced. Given these results and the fact that there has been relatively little application of EO to combinatorial problems, there exists scope to extend it while still retaining its fundamental characteristics. This section describes modifications and enhancements to EO that make it better able to solve assignment type problems.

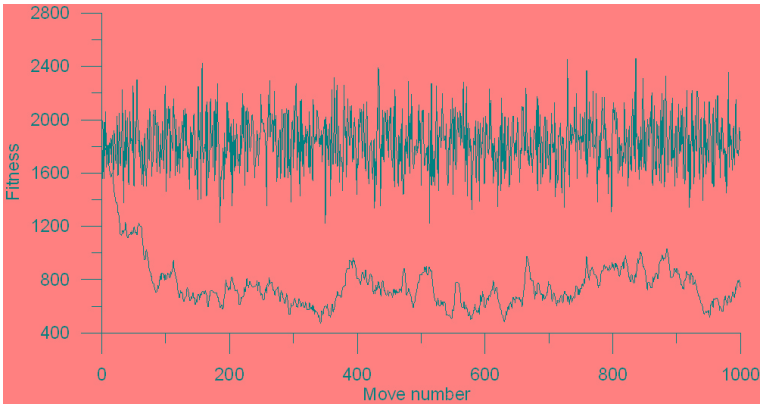

Fig. 10. The fitness of the single solution during the steps of the EO algorithm (lower trace) compared with one thousand randomly chosen solutions. Note the upper trace is not that of a random hill climber.

There are three important elements that EO potentially needs to help it become competitive with more established meta-heuristics. These are a) transition operators and constraint handling techniques, b) a population framework and c) local search.

6.1 Transition Operators and Constraint Handling

The related topics of of transition operators and constraint handling within meta-heuristic search algorithms have been much discussed and explored in the literature. Three broad methods of constraint handling have often been applied:

1. *Use a penalty approach* – Penalise the objective value according to the amount of overall constraint violation.
2. *Allow the solver to search across feasible and infeasible space* – Report the best feasible solution at the end of the search.
3. *Restore feasibility* – Using a special purpose algorithm to transform an infeasible solution to a feasible one.

Method 1 has been used in many meta-heuristic implementations with mixed success. The major difficulty is to determine the best form and weightings of the penalties. A recent penalty based EO [19] showed that it is nearly, but not quite, competitive with state-of-the-art heuristics.

Methods 2 and 3, however, can easily be applied to EO, with Method 2 particularly taking advantage of the natural EO algorithm (as shown in Randall and Lewis [31] and Randall [29]). As EO only makes a small change at each iteration – allowing many transitions to be made in a computationally reasonable time – it may not matter that the solution is feasible at all times. Overall, many feasible solutions will be potentially produced depending on the difficulty of the constraints. Increasing the proportion of feasible solutions may be accomplished by the use of

the partial feasibility restoration algorithm (discussed next). At each iteration of the algorithm, the feasibility status of the solution is determined. The choice of solution component, using EO's rules, is as follows for two of the test problems:

- *A feasible solution* – An EO move is performed to optimise the objective function. A move changes a poor solution component value to a random one. For the GAP, a poor assignment of an agent to a job is replaced by the assignment of that job to a random agent. However, for the BPP, the objective is to minimise the number of groups, therefore a random item is taken from a relatively small sized bin and reassigned to a random one.
- *An infeasible solution* – The focus changes to moving the solution back to a feasible state. As such, a component value to change is chosen according to the amount of infeasibility it contributes to the solution. This will vary from problem to problem. The specifications for each problem are:
  - GAP: This is given as the amount of resources required for a job assigned to an agent. Only overloaded/infeasible agents are examined. The job assignments with resource requirements that most closely match the the amount of agent infeasibility are more likely to be chosen.
  - BPP: A light item from an overfull bin is most likely to be selected. This is the case as it is easier to move light items to fill the spare capacity of other bins. A number of such moves will decrease its infeasibility.

The CSAHLP is a different matter. Based on the work of Randall [30], there exists a node allocation heuristic and feasibility restoration algorithm. The former will best allocate non-hub nodes to hub nodes. If this fails, the latter will restore feasibility.

Partial Feasibility Restoration

It may take a considerable number of EO iterations to move from infeasible space to feasible space using the methods mentioned above. To effectively make use of the available EO iterations, a general purpose heuristic can be developed to reduce the amount of infeasibility of a solution (the prototype for which is given in Randall [29]). Partial feasibility restoration is a simple, non-degenerative, parameter-free process. In some ways, it resembles standard local search, except it tries to minimise infeasibility rather than optimise the problem's objective. It is represented by Algorithm 3. Note that it is applicable across ATPs and does not guarantee that a feasible state will result. It is an $O(MN)$ algorithm where M is the number of groups, and N is the number of items. This algorithm will be used for the GAP and BPP. As mentioned previously, CSAHLP uses its own full feasibility restoration algorithm.

The amount of contribution of infeasibility of an item within a group is, naturally enough, calculated differently for different problems. In the case of BPP, it is an item whose weight most nearly matches the amount of overloading for a bin. For the GAP, the item is simply the one whose resource requirement covers the amount of the agent's infeasibility.

6.2 A Population Model

A simple population extension mechanism to EO was presented by Randall [29]. In it, a fixed number of solutions/individuals form a population. At preset intervals throughout the execution of the search, a population interaction would occur.

Algorithm 3 Generalised partial feasibility restoration for assignment type problems.

```
for all groups do
  if this group is infeasible then
    Determine the item within the group whose resource requirement most closely
    matches the group's amount of infeasibility
    Find a new group that can take this item without itself becoming infeasible
    if such a group exists then
      Update the solution, its cost (if any) and the amount of its infeasibility
    else
      Do nothing
    end if
  end if
end for
```

This interaction identified the worst member of the population (as measured by the objective function) and deleted it. Besides it, the two nearest neighbours (in terms of common solution component values) were also removed in accordance to the Bak-Sneppen model. Three new solutions were randomly generated, added to the population, and the search resumed.

While the above approach produced statistically significantly superior results to a standard implementation on GAP instances, its main drawback was that it required the user to specify the number of times interactions would occur throughout the search process. It would be better if the system could calculate this information for itself based on dynamic information about the search.

As EO does not converge, solution similarity between the population members would not be a natural cause to trigger an interaction. It is better to look for a population in which solution qualities widely diverge. This indicates that the least fit members should be eliminated. The equivalent in Nature would be a cull of the weak of a population so as to maximise the overall chances of a species' survival. The probability that an interaction will occur at an iteration can be calculated according to Equation 2. Note as well that the probability increases as the iterations pass since the last population interaction.

$$p = \left(1 - \frac{cost(best)}{cost(worst)}\right)^{1/l} \tag{2}$$

Where:

l is the number of iterations since the last population interaction occurred.

In terms of the solution replacements, a small augmentation of Randall [29] is proposed here. Instead of replacing all three solutions (i.e., the worst solution and its two closest neighbours) with random solutions, one of these is now a copy of the best found solution. This balances the need for diversity of solutions against that of exploration around the best (known) solution. Again, as EO does not converge, there is no danger of creating a population of similar, stagnating solutions. Rather, it will allow EO to search more extensively in good neighbourhoods.

6.3 Local Search

Meta-heuristic search algorithms, on the whole, are capable of coarse grain search. In essence, they provide very good starting points for fine grained searching known as *local search*. This point was demonstrated for EO in Hendtlass and Randall [20]. As such, local search is solely driven toward optimising the objective function, moving only in feasible space, and making purely greedy moves. However, it is useful for obtaining locally optimal solutions, which standard EO does not guarantee. Local search can be performed each time a feasible solution is produced.

The details of the local search implementation for each problem is given below.

- BPP – The algorithm of Alvim, Aloise, Glover and Ribeiro [1] (as reported in Levine and Ducatelle [23]) initially determines the two least loaded bins. The items from these are moved to a "free list", and the bins removed. Three types of exchange operations are attempted. In the first, two items from the free list are exchanged with two items from a bin (subject to the capacity constraint being satisfied). This process is repeated for all bins and this is attempted for combinations of two bin items for one free item and then one bin item for one free item. After this, all free items that can be fit into existing bins have been. A new bin is created with the left over items. The aim of this procedure is to reduce the number of overall bins by at least one. Even though this algorithm is designed for the BPP, it can equally be applied to problems such as graph colouring. A more detailed description of the algorithm (as well as an example) can be found in Levine and Ducatelle [23].
- GAP – Two effective operators [29] will be used. "Move" moves an item from one agent to another. The job and agent are chosen such that the (negative) change in the objective function is the greatest. This is a variable length search stopping when an improving move cannot be found. "Swap" works in a similar way except that at each iteration, two items are chosen such that their swap will lead to the most improvement in the objective function.
- CSAHLP – There are six local search operators that are appropriate for the CSAHLP [15]. Note that some of the operators discussed below are described in terms of 'clusters'. A cluster refers to a group of nodes that are all assigned to a particular hub.
  1. *Relocate Node* – A node is reassigned to a different hub.
  2. *Swap Nodes* – Two nodes swap the clusters to which they belong.
  3. *Create a New Cluster* – A non-hub node is made a hub node. No nodes (apart from itself) are assigned to this new hub.
  4. *Split a Cluster* – Half of the nodes of a cluster are reassigned to a new cluster. A node from the new cluster is chosen as the hub.
  5. *Merge Clusters* – Two clusters are merged into one. The hub node of one of the clusters becomes the hub node of the new, enlarged cluster.
  6. *Relocate Hub* – The hub of a cluster is reassigned to another node.

  According to Randall [30], the most effective way to apply these operators, at each (EO) iteration, is to first randomly order the transition operators in a list. Each operator is applied, attempting all possible transitions, and all improving moves are kept. This process is continued until an improving move cannot be produced by any of the operators.

7 Computational Experiments

In this section, the effectiveness of EO, along with its support heuristics and population model, is evaluated on benchmark GAP, BPP and CSAHLP problems.

The test problem instances are drawn from three benchmark sets from the literature:

- GAP – These are the large-sized set of Chu and Beasley [11]. They have also been used in the study by Randall [29].
- BPP – The test set of problems from the OR-Library [5] are used. These range in size from 120 to 500 items.
- CSAHLP – These instances are from the benchmark set proposed by Ernst and Krishnamoorthy [15]. A fuller explanation of these problems can be found in Randall [30].

The computing platform used to perform the experiments is a 3GHz Pentium 4 based PC. Each problem instance is run across ten random seeds. The experimental programs are coded in the C language and compiled with `gcc`.

Each problem instance is allowed to run for 500000 iterations. Two sets of results are generated, one for the single solution version and one for the population variant. In terms of the population approach, the total number of iterations is divided amongst the population members, rather than being that number of generations. This helps to ensure a fairer comparison with the single EO versions.

The two other parameters that need to be set are τ and the population size. A value of 1.4 is used for τ as this has been found to give good quality results in a number of previous studies [9, 20, 29, 31] and balances the need for exploration with that of exploitation. Twenty individuals constitute a population in these experiments. However, further investigation will explore the effect of different values of both these parameters.

Tables 2, 3 and 4 show the results for GAP, BPP and CSAHLP respectively. The results are expressed as relative percentage deviations (RPD) from the optimal/best-known cost, i.e., $RPD = \frac{cost}{optimal} \times 100$. Thus a result of 0 corresponds to the optimal/best-known cost. "min", "med" and "max" denote minimum, medium and maximum respectively.

Table 2. The single and population results for the GAP. Note that the first number in the problem name (first column) represents the number of agents while the second is the number of jobs. For instance, 'A5-100' is of Type A with 5 agents and 100 jobs. The single results are reproduced from Table 2 (last set) of Randall [29].

		Single			Population		
Name	Optimal	min	med	max	min	med	max
A5-100	1698	0	0	0	0	0	0
A5-200	3235	0	0	0	0	0	0
A10-100	1360	0	0	0	0	0	0
A10-200	2623	0	0	0	0	0	0
A20-100	1158	0	0	0	0	0	0
A20-200	2339	0	0	0	0	0	0
B5-100	1843	0.71	1.03	1.41	0	0.3	0.33
B5-200	3553	0.42	0.48	0.56	0.06	0.07	0.2
B10-100	1407	0	0	0	0	0	0
B10-200	2831	0.6	0.76	1.02	0	0.11	0.18
B20-100	1166	0.09	0.17	0.26	0	0	0.09
B20-200	2340	0.17	0.21	0.26	0	0.11	0.21
C5-100	1931	0.36	0.6	0.78	0	0.05	0.31
C5-200	3458	0.29	0.52	0.67	0	0.04	0.14
C10-100	1403	0.71	1.1	1.28	0.07	0.14	0.29
C10-200	2814	0.46	0.82	1.03	0	0	0.21
C20-100	1244	0.72	1.05	1.13	0	0.08	0.32
C20-200	2397	0.92	1.17	1.29	0	0	0.13
D5-100	6373	1.37	1.54	1.65	0.25	0.45	0.69
D5-200	12796	1.52	1.63	1.71	0.12	0.27	0.61
D10-100	6379	2.15	2.56	2.76	0.49	0.74	1.47
D10-200	12601	1.24	1.54	1.6	0	0.04	0.13
D20-100	6269	2.14	2.47	2.58	0.33	0.45	1.36
D20-200	12452	1.55	1.69	1.81	0	0.17	0.48

Table 3. The single and population results for the BPP. Note that the problem name (first column) indicates the number of items. For instance, u120_00, is the first problem of the set of problems that have 120 items in each.

		Single			Population		
Name	Optimal	min	med	max	min	med	max
u120_00	48	0	0	2.08	0	0	0
u120_01	49	0	0	0	0	0	0
u120_02	46	0	0	0	0	0	0
u120_03	49	0	2.04	4.08	1.02	2.04	2.04
u120_04	50	0	0	0	0	0	0
u120_05	48	0	0	0	0	0	2.08
u120_06	48	0	0	2.08	0	0	2.08
u120_07	49	0	0	0	0	0	0
u120_08	51	0	0	1.96	0	0	0
u120_09	46	0	1.09	2.17	0	2.17	2.17
u120_10	52	0	0	0	0	0	0
u120_11	49	0	0	2.04	0	0	0
u120_12	48	0	2.08	2.08	0	2.08	2.08
u120_13	49	0	0	0	0	0	0
u120_14	50	0	0	0	0	0	0
u120_15	48	0	0	2.08	0	0	0
u120_16	52	0	0	1.92	0	0	0
u120_17	52	0	0	5.77	0	1.92	1.92
u120_18	49	0	0	0	0	0	0
u120_19	50	0	0	2	0	0	0
u250_00	99	0	0.51	1.01	0	1.01	1.01
u250_01	100	0	0	1	0	0	1
u250_02	102	0	0.49	0.98	0	0.98	0.98
u250_03	100	0	0	0	0	0	1
u250_04	101	0	0	0.99	0	0.99	0.99
u250_05	101	0	0.99	1.98	0.99	0.99	1.98
u250_06	102	0	0	0	0	0	0
u250_07	104	0	0	3.85	0	0	0
u250_08	105	0.95	0.95	0.95	0.95	0.95	1.9
u250_09	101	0	0.99	2.97	0.99	0.99	0.99
u250_10	105	0	0	1.9	0	0.95	0.95
u250_11	101	0.99	0.99	0.99	0.99	0.99	0.99
u250_12	106	0	0	0.94	0	0	0.94
u250_13	103	0	0	0.97	0	0	0.97
u250_14	100	0	0	1	0	1	1
u250_15	105	0.95	0.95	3.81	0.95	1.9	1.9
u250_16	97	0	0	3.09	1.03	1.03	1.03
u250_17	100	0	0	1	0	0	0
u250_18	100	1	1	1	1	1	1
u250_19	102	0	0	0.98	0	0	0.98
u500_00	198	0.51	0.51	2.53	0.51	1.01	1.01
u500_01	201	0.5	0.5	0.5	0.5	1	1
u500_02	202	0	0.5	0.99	0.5	0.74	0.99
u500_03	204	0.49	0.49	1.47	0.98	0.98	0.98
u500_04	206	0	0.49	1.94	0.49	0.49	0.97
u500_05	206	0	0.97	2.91	0.49	0.73	0.97
u500_06	207	0.48	0.72	1.45	0.97	0.97	1.45
u500_07	204	0.49	1.23	3.43	0.98	1.47	1.96
u500_08	196	0	0.26	1.02	0.51	0.51	1.02
u500_09	202	0	0	0.5	0	0.5	0.99
u500_10	200	0	0.5	0.5	0.5	0.5	1
u500_11	200	0.5	0.5	3	0.5	1	1.5
u500_12	199	0.5	0.5	1.01	0.5	1.01	1.01
u500_13	196	0	0.51	1.53	0	0.51	0.51
u500_14	204	0.49	0.49	8.33	0.49	0.49	0.98
u500_15	201	0.5	0.5	0.5	0.5	0.5	1
u500_16	202	0	0	0.99	0	0.5	0.5
u500_17	198	0.51	0.51	0.51	0.51	1.01	1.01
u500_18	202	0	0.5	1.98	0.99	1.49	1.49
u500_19	196	0.51	0.51	1.02	0.51	1.02	1.53

Table 4. The single and population results for the CSAHLP. Note that the problem name indicates the number of nodes in the instance. The cost of 50TT corresponds only to the best known cost. For this problem, the single version of the algorithm could only generate solutions in two of its runs, while the other eight could not generate feasible solutions at all.

		Single			Population		
Name	Optimal	min	med	max	min	med	max
10LL	224250.1	0	0	0	0	0	0
10LT	250992.3	0	0	0	0	0	0
10TL	263399.9	0	0	0	0	0	0
10TT	263399.9	0	0	0	0	0	0
20LL	234691	0	0	0	0	0	0
20LT	253517.4	0	0	0	0	0	0
20TL	271128.2	0	0	0	0	0	0
20TT	296035.4	0	0	0	0	0	0
25LL	238978	0	0	0	0	0	0
25LT	276372.5	0	0	0	0	0	0
25TL	310317.6	0	0	0	0	0	0
25TT	348369.2	0	0	0	0	0	0
40LL	241955.7	0	0	0	0	0	0
40LT	272218.3	0	0	0	0	0	0
40TL	298919	0	0	0	0	0	0
40TT	354874.1	0	0	0	0	0	0
50LL	238520.6	0	0	0	0	0	0
50LT	272897.5	0	0	0	0	0	0
50TL	319015.8	0	0	0	0	0	0
50TT	417441	0	0	0	0	0.06	0.41

As reported in Randall [29], the single solution results in Table 2 were statistically better than a state-of-the-art ACO implementation. In all cases, however, the population approach for the GAP was able to find equivalent or better quality results for all problem instances. The new population results are also superior to the previous population results [29]. This may be attributed to the new model's feature of allowing more concentrated search around best found solutions. This is only practicable because EO will not converge on these solutions again – as would be the case for other meta-heuristics.

For both the single and population versions of EO for bin packing, the algorithms could achieve very good quality results, being at most a few percent away from the optimal results. Unlike the GAP, there is no clear distinction between the single and population results for the minimum results. However, the population version's maximum is less deviant than the other as it is always at most two percent away from the optimal result. These results are very comparable to the ACO implementation of Levine and Ducatelle [23].

The performance of EO on CSAHLP in comparison to ACO [30] was more or less equivalent. Both methods required relatively few iterations to achieve optimal solutions. Much of this may be attributed to the powerful local search heuristics. However, as noted in Randall [30], ACO (and therefore EO) provide a powerful coarse grain optimisation framework. This is evidenced by the fact that a random descent heuristic could not achieve the same level of results. Additionally, the work by Hendtlass and Randall [20] has shown, for bin packing, that pure local search extensions to EO are necessary to produce competitive computational results, despite the computational costs these may incur.

8 Conclusions

EO is a relatively new and simple meta-heuristic that is based on the elimination of poorly performing solution components, rather than the explicit incorporation of necessarily good values. As such it does not converge on certain solutions, giving it some advantages over more traditional meta-heuristics such as GAs and ACO. However, application of the canonical algorithm will often lead to relatively poor performances, this having been demonstrated in this chapter with the bin packing problem (Section 5). In order to lift performance so that EO becomes comparable with other algorithms requires additional support heuristics. Generalised algorithms have been proposed herein that can handle constraints, partially restore feasibility and create and maintain a population of solutions. These combined with local search have indeed shown that EO is capable of producing very good quality solutions.

An area that we are currently investigating is concerned with the management of populations. One of the key questions is that of population size. It may be possible to allow the population to shrink or grow according the progress of the search.

References

1. A. Alvim, D. Aloise, F. Glover, and C. Ribeiro. Local search for the bin packing problem. In *Extended Abstracts of the 3rd Metaheuristics International Conference*, pages 7–12, 1999.
2. P. Bak. *How Nature Works*. Springer, New York, USA, 1996.
3. P. Bak and K. Sneppen. Punctuated equilibrium and criticality in a simple model of evolution. *Physical Review Letters*, 71:4083–4086, 1993.
4. P. Bak, C. Tang, and K. Wiesenfeld. Self-organized criticality: An explanation of 1/f noise. *Physical Review Letters*, 59:381–384, 1987.
5. J. Beasley. OR-Library: Distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41:1069–1072, 1990.
6. S. Boettcher and A. Percus. Extremal optimization: Methods derived from co-evolution. In *Proceedings of the Genetic and Evolutionary and Computation Conference*, pages 825–832. Moran Kaufmann, 1999.
7. S. Boettcher and A. Percus. Combining local search with co-evolution in a remarkably simple way. In *Proceedings of the Congress on Evolutionary Computation*, pages 1578–1584, Piscataway, NJ, 2000. IEEE Service Center.
8. S. Boettcher and A. Percus. Nature's way of optimizing. *Artificial Intelligence*, 119:275 – 286, 2000.

9. S. Boettcher and A. Percus. Extremal optimization for graph partitioning. *Physical Review E*, 64, 2001.
10. S. Boettcher and A. Percus. Extremal optimization: An evolutionary local search algorithm. In H. Bhargava and N. Ye, editors, *Computational Modeling and Problem Solving in the Networked World*, Interfaces in Computer Science and Operations Research, pages 61–77. Kluwer Academic Publishers, 2003.
11. P. Chu and J. Beasley. A genetic algorithm for the generalised assignment problem. *Computers and Operations Research*, 24:17–23, 1997.
12. D. Costa and A. Hertz. Ants can colour graphs. *Journal of the Operational Research Society*, 48:295–305, 1997.
13. 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.
14. J. Duch and A. Arenas. Community detection in complex networks using extremal optimization. *Physical Review E*, 72, 2005.
15. A. Ernst and M. Krishnamoorthy. Solution algorithms for the capacitated single allocation hub location problem. *Annals of Operations Research*, 86:141–159, 1999.
16. E. Falkenauer. A hybrid grouping genetic algorithm for bin packing. Technical Report CP 106 - P4, CRIF Industrial Management and Automation, 50 Av. F.D. Roosevelt, B-1050 Brussels, Belgium, 1994.
17. R. Galski, F. de Sousa, F. Ramos, and I. Muraoka. Spacecraft thermal design with the generalized extremal optimization algorithm. In H. Orlande and J. Colaco, editors, *Proceedings of Inverse Problems, Design and Optimization*, pages 61–75, 2004.
18. D. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison Wesley, Reading MA, 1989.
19. P. Gómez-Meneses and M. Randall. Extremal optimisation with a penalty approach for the multidimensional knapsack problem. In *Proceedings of the Simulated Evolution and Learning Conference*, volume 5361 of *Lecture Notes in Computer Science*, pages 229 – 238, 2008.
20. T. Hendtlass and M. Randall. Extremal optimisation for bin packing. In *Proceedings of the Simulated Evolution and Learning Conference*, volume 5361 of *Lecture Notes in Computer Science*, pages 220 – 228, 2008.
21. T. Kampke. Simulated annealing: Use of a new tool in bin packing. *Annals of Operations Research*, 16:327–332, 1988.
22. J. Kennedy and R. Eberhart. The particle swam: Social adaptation in social information-processing systems. In *New Ideas in Optimization*, pages 379–387. McGraw-Hill, London, 1999.
23. J. Levine and F. Ducatelle. Ant colony optimisation and local search for bin packing and cutting stock problems. *Journal of the Operational Research Society*, 55:705–716, 2004.
24. S. Martello and P. Toth. An algorithm for the generalized assignment problem. In *Proceedings of the 9^{th} International Federation of Operational Research Societies' Conference*, pages 589–603, Hamburg, Germany, 1981.
25. A. Middleton. Improved extremal optimization for the ising spin glass. *Physical Review E*, 69, 2004.
26. I. Moser and T. Hendtlass. On the behaviour of extremal optimisation when solving problems with hidden dynamics. In *Proceedings of the $19^{scriptsizeth}$*

International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, volume 4031 of *Lecture Notes in Artificial Intelligence*, pages 292–301. Springer-Verlag, 2006.

27. I. Moser and T. Hendtlass. Solving problems with hidden dynamics - comparison of extremal optimisation and ant colony system. In *Proceedings of the Congress on Evolutionary Computation*, pages 1248–1255, 2006.
28. I. Moser and T. Hendtlass. Solving dynamic single-runway aircraft landing problems with extremal optimisation. In *Proceedings of the IEEE Symposium on Computational Intelligence in Scheduling*, pages 206–211, 2007.
29. M. Randall. Enhancements to extremal optimisation for generalised assignment. In M. Randall, H. Abbass, and J. Wiles, editors, *The Third Australian Conference on Artificial Life*, volume 4828 of *Lecture Notes in Artificial Intelligence*, pages 369–380, Berlin, 2007. Springer.
30. M. Randall. Solution approaches for the capacitated single allocation hub location problem using ant colony optimisation. *Journal of Computational Optimization and Applications*, 39:239–261, 2008.
31. M. Randall and A. Lewis. An extended extremal optimisation model for parallel architectures. In *Proceedings of the 2nd IEEE International e-Science and Grid Computing Conference*. IEEE Computer Society, 2006. (Workshop on Biologically-inspired Optimisation Methods for Parallel and Distributed Architectures: Algorithms, Systems and Applications).
32. A. Shmygelska. An extremal optimization search method for the protein folding problem: The Go-model example. In *Proceedings of the Companion to the Genetic and Evolutionary Computation Conference*, pages 2572–2579, New York, NY, USA, 2007. ACM.
33. D. Xiaodong, W. Cunrui, L. Xiangdong, and L. Yanping. Web community detection model using particle swarm optimization. In *Proceedings of the Congress on Evolutionary Computation*, pages 1074 – 1079, 2008.
34. T. Zhou, W. Bai, L. Cheng, and B. Wang. Continuous extremal optimization for Lennard-Jones clusters. *Physical Review E*, 72, 2006.