12-1-2010

Intensification strategies for extremal optimisation

Marcus Randall
Bond University, marcus_randall@bond.edu.au

Andrew Lewis
Griffith University

Intensification Strategies for Extremal Optimisation

Marcus Randall[1] and Andrew Lewis[2]

[1] School of Information Technology, Bond University
Queensland, Australia
mrandall@bond.edu.au
[2] Institute for Integrated and Intelligent Systems, Griffith University
Queensland, Australia
a.lewis@griffith.edu.au

Abstract. It is only relatively recently that extremal optimisation (EO) has been applied to combinatorial optimisation problems. As such, there have been only a few attempts to extend the paradigm to include standard search mechanisms that are routinely used by other techniques such as genetic algorithms, tabu search and ant colony optimisation. The key way to begin this process is to augment EO with attributes that it naturally lacks. While EO does not get confounded by local optima and is able to move through search space unencumbered, one of the major issues is to provide it with better search intensification strategies. In this paper, two strategies that compliment EO's mechanics are introduced and are used to augment an existing solver framework. Results, for single and population versions of the algorithm, demonstrate that intensification aids the performance of EO.

1 Introduction

Extremal optimisation (EO) [4] has two main attractions for the researcher and practitioner. The first is that it is very easy to understand and implement. The base algorithm, in comparison to other meta-heuristics, is very simple. This is described in detail in the next section. Perhaps more important is the fact that, unlike its counterparts, it does not converge on a single, or set of, locally optimal solution(s). This avoids the problem of premature convergence. Instead it moves continually through search space, largely being repelled by poor regions of the search space, rather than being specifically attracted to good areas.

The natural, continual diversification of EO's search suggests that work on the algorithm should be devoted to improving its ability to focus on good and promising areas of the search space. This increase in search focus is traditionally referred to as *intensification*, and is present in varying degrees in all meta-heuristic techniques. There exists a reasonably large body of literature for explicit intensification strategies applied to a range of these meta-heuristics, rather than to EO itself. According to Glover and Laguna [10], these approaches can

be grouped into two broad classes. The first set of techniques gathers elite solutions found during the search and allows the meta-heuristic to revisit them when in intensification mode. This is characterised by works of Gambardella, Taillard and Dorigo [9] and Blum [3]. The former propose a hybrid ant system for the quadratic assignment problem (QAP) that is known as HAS-QAP. In an intensification trial phase, the initial solution it uses is the best solution found to date, so it thus corresponds to using one elite solution. Blum [3] follows a similar idea to this, except that the search is allowed to intensify around a set of elite solutions rather than just one.

The other main approach is based on the frequency of incorporation of solution component values[1] into solutions, and their association with quality. If solution components are generally associated with better solutions, it would be prudent to intensify search around a combination of these values. This is exemplified by Randall [11] and Beausoleil [2]. The former work, based on ant colony systems (ACS) (an ant colony optimisation meta-heuristic), analysed the frequency of incorporating components into the colony's solutions. In an intensification phase, frequently incorporated components were highly weighted in the probability selection equations. This is because, according to the ACS rules, they have been associated with good quality solutions in the past. Three search phases (normal, intensification and diversification) were triggered on the basis of the progress of the search. Intensification is performed if the search frequently receives good solutions above some threshold rate, which suggested that the search was in a promising area of the space. Overall, improved solution costs were obtained for larger travelling salesman problem (TSP) instances over a standard ACS strategy. In the latter work [2], parts of the solution structure were kept constant to allow the tabu search mechanism to concentrate on regions containing high quality solution components (as identified through the frequency memory). Applying this strategy, along with frequency-based diversification and path relinking, to the one-machine problem with a weighted tardiness objective produced very good performance in terms of solution quality and time.

The remainder of the paper is organised as follows. Section 2 gives an overview of the mechanics of EO and a description of a framework to help it process constrained problems. Section 3 describes two different intensification methods and how they are integrated with, and augment EO. Using a standard set of generalised assignment problems (GAPs), the intensification-enhanced methods are compared to canonical EO (as well as other algorithms) in Section 4. Finally, the conclusions and future research directions are given in Section 5.

[1] A solution component simply refers to each separate value in the solution vector. In essence, a component represents a building block of the problem. For example, in the generalised assignment problem [8], the assignment of an agent to a job represents a single component.

2 Extremal Optimisation

Boettcher and Percus [4, 6, 7] describe the general tenets of EO. In many ways, it operates counter to other meta-heuristic search algorithms. Instead of actively seeking good solutions through either incremental improvement (such as tabu search and genetic algorithms) or construction (such as ant colony optimisation), bad solutions are actively discouraged. The main advantage of this, as previously mentioned, is that EO will not prematurely converge on a locally optimal solution or a set thereof.

At each iteration of the algorithm, a poor solution component value (as defined by some incremental cost measure) has its value simply changed to a random value (which, of course, is different from the initial value). In the original version of EO, this chosen solution component was always the worst. However, the performance of EO in this form was not favourable. Allowing the component to be chosen probabilistically, according to Equation 1, helped to increase EO's performance and competitiveness with other meta-heuristics.

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

Where: i is the rank of the component, τ is a parameter that alters the probability distribution, P_i is the probability ($P_i = [0, 1]$) that component i is chosen and n is the number of components.

All components are ranked from worst (rank 1) to best (rank n). The set of probabilities are calculated from the ranks and the parameter τ. Essentially, a τ value of 0 gives a random search whereas τ approaching ∞ corresponds to greedy search.

From the above, the general EO (Algorithm 1) is formed.

Algorithm 1 General EO algorithm for a minimisation problem.

1: *best_cost* = Form an initial solution, x, to the problem
2: Calculate the probability, P, for each rank based on the value of τ and n (using Equation 1)
3: **while** the stopping criterion is not met **do**
4: Rank the n solution components from worst to best, based on x and the objective function
5: i = Choose the component value to change using roulette wheel selection (or similar)
6: Assign $x(i)$ a different, random, value
7: *cost* = Evaluate the cost of x according to the objective function
8: **if** *cost* < *best_cost* **then**
9: *best_cost* = *cost*
10: **end if**
11: **end while**
12: Report *best_cost*
13: **end**

2.1 A Framework for Constrained Problem Solution

The following describes three elements that have previously been used [12, 13] to enable EO to solve constrained optimisation problems. The discussion is cast in terms of the problems solved in this paper, the GAP. However, they are sufficiently general to be able to be applied to other problems.

1. *Feasibility Management* – While EO only makes a small change at each iteration - allowing many transitions to be made in a computationally reasonable time - their randomness allows transitions that potentially result in infeasible solutions. Overall, many feasible solutions will likely be 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. This is a simple, non-degenerative, parameter-free heuristic that reduces the amount of infeasibility of a solution. Effectively it helps speed EO back to feasible space, but it does not guarantee that a feasible solution will be produced. As such, it subsequently has a low computational complexity of $O(MN)$ for the GAP, where M is the number of agents and N is the number of jobs.
2. *Population Model* – Unlike many other meta-heuristics, canonical EO only manipulates a single solution. Techniques such as genetic algorithms, ant colony optimisation and particle swarm optimisation derive much of their search power from co-operating and interacting solutions. A simple way to add this extension to EO is to employ some aspects of the Bak-Sneppen model of evolution [1]. Put simply, the weakest member of a population and its two closest neighbours die and are replaced by new members. These new members are two generated at random and one that is a copy of the best found solution. The population interactions are triggered probabilistically according to a divergence measure.
3. *Local Search* – Standard greedy descent search can be applied each time a feasible solution is generated. For the GAP, two operators are possible. "Move" moves a job from one agent to another and "Swap" swaps two jobs (from different agents). This is a variable length search that stops when an improving move cannot be found. These used in combination are very effective and add relatively little to the computational time [12, 13].

Another, new component that can be added to this standard framework is a general method to introduce search intensification. This is described in the next section.

3 Intensification Methods

Intensification schemes that take advantage of the native EO algorithm can be developed using frequency and historic information. Two separate methods that integrate into, and extend, the previously described EO framework are thus developed and outlined below.

3.1 Intensification based on Frequency Information

Over a number of iterations of EO, an association between solution component values and quality of solutions can be built. The solution component values that are generally present in good quality solutions may be worthy of further investigation. Specifically, if these values are fixed temporarily within the search process, it allows EO to better explore the (smaller) region around those values. This may in turn reveal improved solutions.

The association of solution component values and overall solution quality can be kept in a matrix indexed by component number and value. Each cell of the matrix holds the average solution quality that that combination of component and value has received to date. This can be used to calculate the probability of temporarily "locking in" that value. There are numerous, sensible, ways that such a probability can be calculated, with one such given in Algorithm 2.

Algorithm 2 Calculation of the probabilities to lock in solution component values. This assumes a minimisation problem.

1: **for** each solution component c **do**
2: v_c = Choose the lowest/best from the association matrix
3: $p_c = \frac{best\_cost}{v_c}$
4: **end for**
5: Rescale the probability vector, p, between its minimum and maximum value

After an initial period of iterations (typically a few hundred iterations), to allow the association matrix to receive sensible values, EO has the possibility of temporarily locking in solution component values. At each iteration that a feasible solution is produced, each of the probability values are compared against uniformly distributed random numbers to determine which combinations of component and values should be locked in. This works with EO's ranking system in a novel way. If a component is locked, it is given the highest available EO rank. In reality this is a "soft" lock, as EO may, in an unlikely case, choose this component value to change. In the event that all components become locked, this protects the system from not being able to make a transition (see step 5 of Algorithm 1).

The search, of course, will quickly stagnate if components are locked for an excessive number of iterations. Conversely, if the period is too short, EO will not have sufficient opportunity to explore the search region bounded by the locked values. The system allows this choice to be a user-defined parameter. Once this number of iterations has expired, the components are then freed. The alternative to fixing this period would be to explore self-adaptation governed by sufficient solution improvement.

3.2 Intensification based on Historic Information

During the course of EO's search, it will typically find many solutions that have objective costs that are the best to date. These solutions, apart from being comparatively good, may indicate that the surrounding space may also contain other solutions of similar, and potentially superior, quality. However, canonical EO will continue through the space without adequately exploring such regions.

The best solutions found during the course of the search can be saved in a specially created archive. EO can revisit these solutions by selecting them from the archive to substitute for poor solutions to be discarded. It is extremely unlikely that EO will follow the same search trajectory after one of the archived solutions is reinstated, since further probabilistically-chosen bad solutions will be replaced with random values. Thus, the region around an archived solution may be better explored by further sampling. The questions that naturally arise in terms of its implementation are: a) when should this intensi cation be activated in the search process; b) how many solutions should constitute the archive and c) how should they be chosen to be used?

Intensification need only be activated when the search process appears to be unable to find improved solutions for an extended period. The current region of search space likely does not contain better solutions than received previously; thus, returning to one of the archived solutions would be more profitable. There are many possible models that could provide this intensification "trigger" mechanism. One such interesting way, given in Equation 2, calculates a probability based on the number of iterations since the last solution improvement was received. As the number of iterations since the production of this solution increases, the probability of activating intensification geometrically increases.

$$p = N \times \frac{I}{T} \tag{2}$$

Where: p is the probability that intensification will be activated, N is the number of non-improving moves since the last improved solution was received, I is the number of (approximate) intensification periods in the search, and is a user definable parameter and T is the total number of iterations.

A fixed-size archive is a simple and practical way of storing the solutions to which search may be returned. There is no way of accurately determining this size *a-priori*, thus it becomes another user defined parameter. As new improved solutions are obtained, and the archive is at capacity, solutions in it will need to be replaced. An effective way to do this is to replace the current worst solution in the archive. As only ever improving solutions are added to the archive, the replaced solution will also be the oldest member.

4 Computational Experiments

The computer used to perform the experiments was a 3 GHz Pentium 4-based PC. Each problem instance was run across ten random seeds. The experimental programs were coded in the C language and compiled with `gcc`. The only two

standard EO parameters were τ and the number of iterations for which it was to be run. τ was set as 1.5 (a value consistent with Boettcher and Percus [5]), and the latter as 500,000. This value is the same as used by Randall [12] and Randall et al. [13]. Results are reported as relative percentage deviations (RPDs) from the known best solution cost for each problem instance. Formally this is given as $\frac{a-b}{b} \times 100\%$ where a is the obtained cost and b is the best known cost.

To test the practical performance of the two intensification schemes, instances of the generalised assignment problem were used. This is because it has now been extensively trialled with EO [12, 13] and as such, new results can be sensibly compared. In addition, it is an $\mathcal{NP}$ hard problem that also has constraints. The test suite of problems was the large-sized set of Chu and Beasley [8]. Further explanations of these may also be found in Randall [12] and Randall et al. [13]. The B, C and D type problem instances were considered here.

4.1 Experiments and Results

In the development of any new algorithm, there will always be design choices and new parameters introduced. It is not possible to test all combinations of algorithm variations and parameter values to determine the best. However, some investigation and analysis is required.

For both of the intensification-enhanced EO variants, the trigger points could be infinitely varied. This is perhaps matter for a more extensive study of the subject than this paper provides. There is one parameter to vary for the Locked version and two for the Archive version. These are as follows:

- *Locked* – A substantial number of iterations should be used to give EO time to explore the region around a solution that has one or more of its components locked. As such, values from the set $\{100, 200, 500, 1000, 5000\}$ iterations were tested.
- *Archive* – The two parameters are size of the archive and the approximate number of intensification periods (see Equation 2). Both were tested with values drawn from the set $\{5, 10, 20, 50, 100\}$.

To test the effect of these parameters, the instances D5-100 and D10-200 were chosen. Both of these problems are relatively hard, yet representative of this assignment problem. Table 1 shows the results of varying the Locked parameter on the canonical and population variants respectively. Due to the sheer size of the Archive results only those for the canonical algorithm for D5-100 are shown here. The Kruskal-Wallis statistical procedure was used to determine if there were any substantive differences between the parameter values. The following was found:

- *Locked* – The single and population versions of the algorithm yielded different results as is evident from the tables. Neither Kruskal-Wallis test detected significant differences amongst the locked period values. Looking at Table 1 shows that, in terms of the median and maximum values, 500 and 5000 iterations were best for the single and population versions of the algorithm respectively.

– *Archive* – Given the combination of two parameters, there are 25 separate cases. These results (for D5-100) are shown in Table 2. D10-200 shows a similar pattern. In the canonical/single version of EO, a significant difference was detected. The best results were achieved using a small archive size (5) and a large number of intensification periods which is confirmed by inspection of the table. This was not the case for the population version. There was no significant difference between the Kruskal-Wallis ranks also verified by the contents of the table. However, a larger archive size (10 members) was better and was used subsequently.

Table 1. Parameter variation results for the Locked version.

		Single			Population		
Instance	Locked Iterations	Min	Med	Max	Min	Med	Max
D5-100	100	0.85	1.26	1.33	0.28	0.75	1.26
	200	0.55	1.18	1.38	0.41	0.86	1.22
	500	0.89	1.2	1.29	0.27	0.77	1.22
	1000	0.93	1.26	1.47	0.5	0.73	1.27
	5000	0.86	1.26	1.41	0.31	0.62	1.22
D10-100	100	2.45	2.93	3.12	1.38	1.83	2.49
	200	2.4	2.97	3.19	1.62	2.04	2.71
	500	2.49	2.93	3.08	1.54	2.03	2.71
	1000	2.53	3.1	3.22	1.51	1.84	2.71
	5000	2.57	3	3.2	1.55	1.83	2.71

Table 2. Parameter variation results for the archive version using D5-100.

	(Size, Periods)									
Descriptor	Single					Population				
	(5,5)	(5,10)	(5,20)	(5,50)	(5,100)	(5,5)	(5,10)	(5,20)	(5,50)	(5,100)
Min	0.82	0.44	0.63	0.58	0.44	0.08	0.24	0.19	0.05	0.05
Med	1.04	0.82	0.81	0.81	0.71	0.35	0.33	0.42	0.29	0.29
Max	1.21	1.11	1.1	1.13	0.82	0.63	0.75	0.67	0.77	0.77
	(10,5)	(10,10)	(10,20)	(10,50)	(10,100)	(10,5)	(10,10)	(10,20)	(10,50)	(10,100)
Min	0.82	0.63	0.63	0.77	0.71	0.09	0.25	0.13	0.16	0.16
Med	1.08	1	0.91	0.86	0.85	0.39	0.43	0.36	0.28	0.28
Max	1.27	1.46	1.37	0.97	1.02	0.78	0.58	0.56	0.52	0.52
	(20,5)	(20,10)	(20,20)	(20,50)	(20,100)	(20,5)	(20,10)	(20,20)	(20,50)	(20,100)
Min	0.77	0.77	0.88	0.72	0.72	0.17	0.08	0.16	0.19	0.19
Med	1.28	1.25	1.11	1.15	0.97	0.29	0.34	0.42	0.34	0.34
Max	1.46	1.43	1.38	1.4	1.05	0.55	0.64	0.74	0.67	0.67
	(50,5)	(50,10)	(50,20)	(50,50)	(50,100)	(50,5)	(50,10)	(50,20)	(50,50)	(50,100)
Min	0.77	0.77	1.05	0.86	0.83	0.02	0.22	0.09	0.27	0.27
Med	1.27	1.25	1.17	1.17	0.93	0.28	0.33	0.29	0.38	0.38
Max	1.46	1.43	1.43	1.4	1.11	0.53	0.55	0.5	0.55	0.55
	(100,5)	(100,10)	(100,20)	(100,50)	(100,100)	(100,5)	(100,10)	(100,20)	(100,50)	(100,100)
Min	0.77	0.77	1.05	0.86	0.83	0.19	0.22	0.28	0.16	0.16
Med	1.27	1.25	1.17	1.17	0.93	0.31	0.35	0.41	0.38	0.38
Max	1.46	1.43	1.43	1.4	1.11	0.94	0.49	0.61	0.63	0.63

Table 3. The cost results received by the Locked and Archive.

	Unintensified		Locked						Archive					
	Single	Pop.	Single			Population			Single			Population		
Instance	Med	Med	Min	Med	Max	Min	Med	Max	Min	Med	Max	Min	Med	Max
B5-100	1.03	**0.3**	0.81	1.47	1.74	0.54	1.33	1.84	0.27	0.33	0.65	0.22	0.33	0.54
B5-200	0.48	**0.07**	0.17	0.37	0.62	0.2	0.45	0.65	0.17	0.24	0.34	0.03	0.08	0.2
B10-100	**0**	**0**	0	**0**	0.14	0	**0**	0.07	0	**0**	0	0	**0**	0
B10-200	0.76	**0.11**	1.41	1.59	2.23	0	0.49	0.64	0.21	0.37	0.53	0	0.12	0.32
B20-100	0.17	**0**	0.17	0.17	0.43	0.09	0.17	0.43	0	0.04	0.17	0	**0**	0.17
B20-200	0.21	0.11	0.09	0.26	0.47	0.09	0.19	0.34	0.04	0.11	0.17	0	**0.06**	0.13
C5-100	0.6	0.05	0.05	0.47	0.62	0.62	0.88	1.19	0.05	0.21	0.47	0	**0**	0.1
C5-200	0.52	0.04	0.23	0.36	0.69	0.29	0.45	0.52	0.17	0.26	0.32	0	**0.03**	0.2
C10-100	1.1	0.14	0.64	1.46	2.35	0.29	0.96	1.21	0.07	0.36	1.14	0	**0.07**	0.21
C10-200	0.82	**0**	1.28	1.6	1.85	0.18	0.44	0.82	0.14	0.25	0.57	0	**0**	0.11
C20-100	1.05	**0.08**	1.45	2.65	3.7	0.48	0.76	1.13	0.08	0.24	1.05	0.08	0.12	0.48
C20-200	1.17	**0**	2.29	2.84	3.5	0.13	0.58	1	0.21	0.5	0.92	0	0.08	0.33
D5-100	1.54	0.45	0.89	1.2	1.29	0.89	1.3	1.68	0.44	0.71	0.82	0.16	**0.28**	0.52
D5-200	1.63	0.27	1.33	1.61	1.79	0.69	0.91	1.36	0.42	0.7	1.01	0.12	**0.24**	0.42
D10-100	2.56	**0.74**	2.4	2.84	2.98	2.01	2.42	2.62	1.07	1.43	1.91	0.66	0.94	1.3
D10-200	1.54	0.04	1.75	2.08	2.31	0.66	1.12	1.72	0.18	0.5	0.96	0	**0**	0.22
D20-100	2.47	**0.45**	3.01	3.4	3.45	1.72	2.15	2.63	0.72	1.45	1.9	0.37	0.68	1.23
D20-200	1.69	0.17	2.63	2.91	3.12	1.25	1.74	2.05	0.52	0.8	1.29	0.06	**0.13**	0.4

Table 3 shows the results for the Locked and Archive versions of the algorithm (in single and population mode). Visual inspection indicates that the Archive version clearly outperforms the Locked version. On all problem instances, for both single and population versions, the former yields solution values with lower RPDs. In more than half the test cases, the population implementation of the Archive version finds the best known solution at least once. On several test cases it finds the best known solution in more than half of the trials, a result unmatched by any other version.

Consistent with the results of previous studies [12, 13], the population model is a more efficient way of running EO. Comparison to the work of Randall et al. [13] shows that (using the Archive version), intensification helps the single version of EO dramatically. On all test cases, it gives improved or better solution costs in terms of minimum, median and maximum values. In the majority of cases, the RPDs achieved have been at least halved. Interestingly, the performance of intensification in the population version of Archive is very similar to the results of Randall et al. [13]. This simply confirms that the population mechanism is a very powerful way of managing EO searches. However, it may be noted that, on the majority of test cases, intensification has yielded lower median RPDs than the use of the population mechanism alone.

5 Conclusions

From the results presented in this paper it is evident that explicit intensification techniques for EO can improve its ability to find good quality solutions. Two relatively simple forms of intensification, based on the notions of locking in solution components with good values, and revisiting previously known good

solutions were investigated. These strategies were enhanced with heuristics that made them more suitable for use with, and contextually applicable to, EO. In particular, the Archive version produces very good solutions. We plan to investigate further refinements on these schemes for a wider range of problems in the near future.

References

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

[2] R. Beausoleil. Intensification and diversification strategies with tabu search: One-machine problem with weighted tardiness objective. In *Proceedings of the Mexican International Conference on Artificial Intelligence*, volume 1793 of *Lecture Notes in Artificial Intelligence*, pages 52–62, 2000.

[3] C. Blum. ACO applied to group shop scheduling: A case study on intensification and diversification. In M. Dorigo, G. Di Caro, and M. Sampels, editors, *Third International Workshop on Ant Algorithms, ANTS 2002*, volume 2463 of *Lecture Notes in Computer Science*, pages 14–27, Brussels, Belgium, 2002. Springer-Verlag.

[4] 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.

[5] 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.

[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 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.

[8] P. Chu and J. Beasley. A genetic algorithm for the generalised assignment problem. *Computers and Operations Research*, 24:17–23, 1997.

[9] L. Gambardella, E. Taillard, and M. Dorigo. Ant colonies for the quadratic assignment problem. *Journal of the Operational Research Society*, 50:167–176, 1999.

[10] F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, Boston, MA, 1997.

[11] M. Randall. A systematic strategy to incorporate intensification and diversification into ant colony optimisation. In H. Abbass and J. Wiles, editors, *Proceedings of the Australian Conference on Artificial Life*, pages 199–208, Canberra, Australia, 2003.

[12] 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.

[13] M. Randall, T. Hendtlass, and A. Lewis. Extremal optimisation for assignment type problems. In A. Lewis, S. Mostaghim, and M. Randall, editors, *Biologically-inspired Optimisation Methods: Parallel Algorithms, Systems and Applications*, volume 210 of *Studies in Computational Intelligence*, pages 139–164. Springer-Verlag, 2009.