How Hard should we Run?

Trading off Exploration and Exploitation in Evolutionary Algorithms for Dynamic Optimisation Problems

Yun-Geun Lee
Seoul National University
Structural Complexity Laboratory
Building 302, Gwanangno 599,
Seoul 151744, Korea
ey9ey9@gmail.com

Bob McKay
Seoul National University
Structural Complexity Laboratory
Building 302, Gwanangno 599,
Seoul 151744, Korea
rimsnucse@gmail.com

ABSTRACT

All evolutionary algorithms trade off exploration and exploitation in optimisation problems; dynamic problems are no exception. We investigate this trade-off, over a range of algorithm settings, on dynamic variants of three well-known optimisation problems (One Max, Royal Road and knapsack), using Yang's XOR method to vary the scale and rate of change. Extremely exploitative algorithm settings performed best for a surprisingly wide range of problems; even where they were not the most effective, they still performed competitively, and even in those cases, the best performers were still far more exploitative than most would anticipate.

Track: Genetic Algorithms

Categories and Subject Descriptors

G.1.6 [**Mathematics of Computing**]: Optimization—*Stochastic programming*

General Terms

Performance

Keywords

Dynamic Optimisation Problem, Evolutionary Dynamic Optimisation

1. INTRODUCTION

This paper originated in some preliminary experimental work comparing selection operators in evolutionary algorithms applied to dynamic optimisation problems (DOPs). In that research, we repeatedly found that high stringency of selection was the most important characteristic – that for most settings over a range of tunable discrete DOP benchmarks, the most stringent selection settings available gave the best performance.

Stringency of selection is an important determiner of the exploration/exploitation trade-off in evolutionary algorithms: the more stringent the selection, the more exploitative is the algorithm – the less it explores. Hence the question naturally arose, just how exploitative could an algorithm be, and still perform well in these discrete DOPs? This research is an attempt to answer this question, by comparing the performance of a variety of different algorithms with differing trade-offs of exploration and exploitation over a range of different tunings of the discrete DOPs. To foreshadow the somewhat surprising results, over a broad range of settings the best performance was obtained by near-maximally exploitative algorithms. In only a very narrow range of settings was limited exploration of value.

The remainder of this paper describes the experiments and discusses the results. Section 2 introduces DOPs and DOP benchmarks, provides some background in dynamic evolutionary algorithms, and the trade-off between exploration and exploitation. In section 3 and 4, we describe our experimental methods and detailed settings, while section 5 presents the results. Section 6 draws out the implications of these results, while section 7 delineates the assumptions and limitations of our work, presents our overall conclusions, and discusses how this work may be carried forward.

2. BACKGROUND

In this section, we briefly introduce dynamic optimisation, discussing some of the special issues which arise in comparing dynamic optimisation algorithms, and previous research about the trade-off between exploration and exploitation.

2.1 Dynamic Optimisation

A dynamic optimisation problem (DOP) is an optimisation problem in which the objective function and/or the environment change over time. A dynamic optimisation problem can be tackled by successively solving a series of static/stationary optimisation problems, for each of which the objective function is kept fixed during the optimisation process. In other words, any algorithm A for solving a static optimisation problem P can easily be extended to solve a dynamic version of P by simply restarting A whenever a change is detected (or in the case of continuous change, sufficiently frequently). However such an approach is generally not a good choice in practice, because it wastes information collected in previous stages of optimisation (before the change happened), especially when the amount of change is limited.

Evolutionary algorithms (EAs), initially designed for static optimisation problems, have been extended to handle DOPs and demonstrated to be useful [3]. One of the main challenges in designing EAs to tackle DOPs is to maintain the adaptability of the algorithms. To an even greater extent than for static optimisation, a dynamic evolutionary algorithm should focus not only on convergence toward the optimal solution, but also on adaptability to change, by maintaining diversity in its population of candidate solutions. However the two requirements of diversity maintenance and convergence are in conflict, so an adaptive EA for solving DOPs has to balance them. A comprehensive review of EAs for solving DOPs and related problems can be found in [3].

2.2 The Trade-off between Exploration and Exploitation

Yang and Tinos [14] previously investigated the effect of selection pressure in DOP. They tested tournament selection of diverse size in standard GA(SGA), as well as restart-SGA(RSGA), which re-initialises the evolution whenever the environment changes. They also compared with a number of new mechanisms, controlling selection pressure adaptively. These new mechanisms appear valuable for DOP. However we found some of their results quite confusing. Specifically, in the case of SGA with tournament selection, we see that when the environmental change is small, in the easiest problem (One Max), tournaments of size 2 performed best. In the more difficult Royal Road problem, large tournaments (size 10) were best for almost all cases; finally, for the most difficult (knapsack) problem, the best performance came from intermediate-sized tournaments (size 6). These results are sufficiently inconsistent, both with each other and with reasonable expectations (one might naively expect that a highly exploratory setting such as tournament size 2 would not be well matched to a simple problem such as One Max with small environmental changes), that we felt that other factors might be at play. Our aim, in this work, is to extend Yang and Tinos' results by investigating other determinants of the exploration/exploitation trade-off, in the hope of teasing out these influences.

Yang further explored the trade-off between exploration and exploitation in [13], where he introduced a number of new DOP methods. However those comparisons made use of roulette wheel selection, which in our experiments generally performs poorly, being insufficiently exploitative.

3. METHODS

3.1 Benchmarking Dynamic Optimisation

The difficulty of dynamic optimisation problems may vary in two main ways. Firstly, the difficulty of static snapshots of the problem may vary – that is, the fitness landscape may be smooth or rugged, low or high dimensional. Second, the change in the fitness landscape may be slow and steady, or rapid and abrupt. Benchmark suites need to incorporate variation in both these aspects.

The variation in the fitness landscape may be either continuous or episodic. It might seem natural to use continuous benchmarks, since many real-World problems exhibit continuous variation in the objective. However continuous variation makes it difficult to disentangle the effects of variation from the effects of fitness landscape roughness. So discrete variation is generally used, as it is thought to give greater insight into the performance of the dynamic optimisation algorithm. With episodic variation, a new issue intrudes: the change in fitness landscape now has two dimensions, the extent of change in each episode, and the frequency of change. So benchmarking needs to study variation in both.

Yang [12, 13, 15] has recently provided a general method for deriving suitable dynamic benchmarks suites from static ones, that can be tuned in both the rate and the scale of change in the objective. It provides a means to convert any static optimisation problem using a binary chromosome into a dynamic optimisation problem, in which both the extent and frequency of change in the objective can be tuned. Each snapshot of the dynamic problem corresponds to a (suitably transformed) version of the underlying static problem (hence may be tuned by choice of the static problem). We detail this method further in subsection 3.3

3.2 Test Problems

Comparing static evolutionary algorithms [10] is a difficult problem. Benchmark biases in optimum location, axial or directional structure, decomposability, rotational invariance, regularity of structure, and scale can all differentially affect the performance of different algorithms [6]. This is unavoidable in principle [11]: there can be no universally-applicable benchmark test. Thus it is important to allow for the inevitable biases in specific benchmark problems, and to study the performance of algorithms over a range of different problem regimes, so as to delineate which algorithms are best suited to which regimes.

For this reason, we used three test problems – not just one – chosen to represent a range of difficulty, from relatively easy to hard. The test problems are detailed below. All problems used a 100-bit binary representation.

One-Max Problem.

The aim in the One-Max (or Bit-Counting) problem [9] is to maximize the number of ones in a binary chromosome with $x_i \in \{0, 1\}$. The fitness of a chromosome is thus naturally defined to be the number of ones in the chromosome. In our experiment, the optimal value was 100.

$$f(\underline{x}) = \sum_{i=1}^{100} x_i \tag{1}$$

Royal-Road Problem.

The Royal-Road problem [7] is one of the most popular GA test functions. Its fitness landscape is reasonably difficult, but easy to comprehend, so that GA performance on it can be relatively easily understood. The problem representation consists of a bit-string which is divided into N non-overlapping blocks, each with K bits. Its total length is therefore NK bits. In this study, the problem consisted of 25 contiguous 4-bit building blocks (i.e. N=25, K=4). Each building block contributed 4 to the fitness if all of its four bits were set to one; otherwise, it contributed 0. Thus the optimum value was 100.

$$f(\underline{x}) = \sum_{i=1}^{25} \bigwedge_{j=1}^{4} x_{i*j} \tag{2}$$

Knapsack Problem.

In a knapsack problem [1], the task is to pack items into a knapsack. The items must be chosen from a pre-defined pool, the weight and profit value for each item being pre-specified. The maximum weight the knapsack can hold is also specified. The aim is to fill the knapsack in such a way as to maximise the profit.

In our version, there were 100 items, with the weight and profit of each item being randomly created in the range $[1 \ldots 32]$. The capacity of the knapsack was set to $\frac{1}{2}$ of the total weight of all items. The fitness of a chromosome was the sum of the profits of the selected items, so long as the weight was within range. However if a chromosome overfilled the knapsack, its fitness was set to the difference between the total weight of all items and the weight of the selected items, multiplied by a small factor 10^{-5}, as a penalty.

3.3 Testing the Effect of Dynamism

If benchmarking is difficult for static optimisation, it is even more complex for dynamic. To variations in function landscape complexity we must now add variations in the rate and scale of change of the optimisation objective [4]. To date, there is little in the way of formal analysis – or even definition [8] – of dynamic optimisation, and even less of its benchmarking. Nevertheless Yang's recent method [13] has rapidly gained acceptance as a benchmarking method, and we follow it here.

Briefly, Yang's benchmark works by using a binary genotype representation. Before the representation is transformed to the final phenotype, it is first XORed with a mask. That is, $f(x, k) = f(x \oplus M(k))$. The mask is changed every τ generation. At a particular epoch k, the previous mask $M(k)$ is updated using a new randomly generated mask $T(k)$, in which $\rho * \text{length}(M(k))$ of the bits are set to 1. The initial mask, $M(1) = \bar{0}$, being the zero matrix, does not change the underlying bitstring. As a result, the first problem is just the underlying static optimisation problem. The subsequent problems are generated by

$$M(k+1) = M(k) \oplus T(k+1) \tag{3}$$

Thus at each epoch, of the bits in the genotypic representation of the phenotype, a fraction ρ change sign. This happens every τ generations.

In this formulation, parameter τ controls the rate of change – the number of generations per epoch. Parameter ρ controls the distance moved in the function domain, and thus is a surrogate for the extent of change in the fitness landscape. The precise effect of ρ will also depend on the relationship between the objective and the domain – symmetries in the objective may reduce the effect of changes in $M(k)$.

4. EXPERIMENTS

4.1 Experimental Methodology

Our primary objective was to investigate the impact of different exploration/exploitation trade-offs on the behaviour of evolutionary algorithms in dynamic optimisation (and in light of our earlier experience, to focus on the more exploitative end of the spectrum). In our experiments, we first compared different selection mechanisms in the context of a baseline – the "Standard" Genetic Algorithm (SGA). However SGA is believed to perform rather poorly on dynamic problems, so we also studied the Random Immigrant GA (RIGA) [2], the elitism-based immigrants GA (EIGA) [13] and the hybrid immigrants GA (HIGA) [13].

We followed the approach of Yang et al. [12, 13, 15] for comparing dynamic optimisation methods and parameter settings, as summarised in subsection 3.3. Like them, we based our tests on a range of static problems of different degrees of difficulty, as described in subsection 3.2, and in sweeping the ρ and τ parameters (subsection 3.3) across a range of possible combinations, to determine the 'sweet spots' for the different combinations of specific GAs with different exploration/exploitation trade-offs. We also followed them in using slight variants of a classic (P, P) algorithm, in which P parents generate exactly P children, by first applying crossover at the specified rate, then applying mutation to the potentially crossed-over children. However we varied the selection operator used in choosing the parents, and the immigration policy (random and/or elite) as determiners of the extent and kind of exploration. Algorithm 1 shows a generic pseudo-code for all these algorithms.

Algorithm 1 Pseudocode for GA of Experiments

```
t ← 0
ImmigrantSize ← RandomImmigrantSize + EliteSize
                                        ▷ Population Initialisation
for i ← 1 to PopSize do
    Pop[0, i] ← RandomIndividual()
end for
Sort Pop[0] by Fitness
while $t < t_{\max}$ do
    t ← t + 1
                                        ▷ Main GA Update Loop
    for i ← 1 to PopSize do
        parent[1] ← Selection(Pop[t − 1])
        parent[2] ← Selection(Pop[t − 1])
        if Random() < CrossoverProbability then
            Cross(parent[1],parent[2],Pop[t, i])
        else
            Pop[t, i] ← parent[1]
        end if
        Mutate(Pop[t, i], MutationProbability)
    end for
    Sort Pop[t] by Fitness
                                ▷ Import Elite Immigrants (if any)
                              ▷ Elite Immigrants replace Least Fit
    for i ← 1 to EliteSize do
        Pop[t, Popsize − i + 1]
        ← Mutate(Pop[t − 1, i], MutationProbabiity)
    end for
                              ▷ Import Random Immigrants (if any)
                            ▷ Random Immigrants replace Least Fit
    for i ← 1 to RandomImmigrantSize do
        Pop[t, Popsize − EliteSize − i + 1]
        ← RandomIndividual()
    end for
    Sort Pop[t] by Fitness
end while
```

The total number of evaluations per generation is PopSize + RandomImmigrantSize + EliteSize, so we held it constant for the different algorithms (SGA, RIGA, EIGA, HIGA).

To summarise the experiments, we varied a number of components of the algorithm to trade off exploration and exploitation at different stages

Table 1: Experimental Parameters

Immigrant Rate†:	**SGA**	**RIGA**	EIGA	HIGA
Random	**0**	**0.2**	0	0.1
Elite	**0**	**0**	0.2	0.1
†Expressed as a proportion of the non-immigrants				
Exploratory:				
Population	**120**	**100**	100	100
τ	**10**, *25*, **50**			
Exploitative:				
Population	**12**	**10**	10	10
τ	**100**, *250*, **500**			
ρ (disruption)	*0.05*, **0.1**, **0.2**, *0.3*, *0.4*, **0.5**, **1.0**			
Epochs	**200**			
Chromosome	**100** bits			
Crossover	Uniform ($p_c = \mathbf{0.6}$; $p_c = \mathbf{0.0}$)			
Mutation	Bit-wise Flip ($p_m = \mathbf{0.01}$)			
Selection				
Roulette				
Tournament	Size **2**,3,4,**5**,6,7,8,9,10,11,**12**			
Truncation	Ratio **0.1**,**0.2**,**0.3**,0.4			
Runs	**50**			

1. Immigration policy (SGA, RIGA, EIGA, HIGA)
2. Selection (Roulette, Tournament, Truncation)
3. Parameters of the selection mechanism (Tournament Size, Truncation Ratio[1])
4. Population size and length of epoch τ (chosen to guarantee a fixed number of fitness evaluations – 1,200, 3,000 or 6,000 – for short, intermediate or long epochs)

The combinations of parameter settings used in these experiments are shown in table 1. We conducted experiments for all combinations of these settings. To meet space requirements, we have limited the presentation of results to those most pertinent to the themes of the paper – these settings were determined post-facto, and are shown in bold in the table. Since the most interesting results relate to the most complex problem (i.e. the Knapsack problem), further combinations of settings are presented for this problem alone; these settings are shown in italic.

The major issue in this work is the level of exploration. At one extreme, with a population of 120, roulette selection and random immigrants, we have a highly exploratory (120,120) algorithm. It could be made even more exploratory by increasing population or incorporating diversity metrics, but as we shall see in the results, there may be little point in looking further in this direction, at least for these problems. At the other extreme, we have a population of 12 with truncation selection of the fittest in an elitist algorithm – i.e. a (1+12) algorithm, a very eager hillclimbing search with virtually no resilience to trapping in local optima.

A key issue is how to measure the performance of a dynamic optimisation algorithm. Should we measure the fitness of the fittest individual at each generation, or average over the population? Should we measure only after the population has adapted to the new optimum, or should our metric include some indication of the performance immediately after the change? Here, the choice ultimately depends on the exact way in which the algorithm will be practically applied, so there is no universal 'right' answer. As a practical expedient, we adopted Greffenstette's offline performance measure [2], known as the best-of-generation (BOG) fitness F_{BOG}, averaged over the N runs, and over the data gathering period, as defined below:

$$\overline{F}_{BOG} = \frac{1}{G}\sum_{i=1}^{G}\left(\frac{1}{N}\sum_{j=1}^{N} F_{BOG_{ij}}\right) \tag{4}$$

where $G = 200 * \tau$ is the total number of generation for each run, $N = 50$ is the total number of runs, and $F_{BOG_{ij}}$ is the best-of-generation fitness of generation i of run j. It is important to note that the offline performance measure $\overline{F}_{BOG}$ is averaged over all generations. Thus it differs from typical performance measures used for static optimisation, which generally use the best fitness of the final generation as a performance measure.

5. RESULTS

In all tables in this section, for ease of interpretation, the best-performing treatment for a particular problem setting (i.e. in a specific column) is printed in large font, and the second best in bold. The column headed XO denotes whether crossover was used (X if it was, - if not).

Table 2: Mean BOG Fitness, One-Max Problem, $\tau = 10$

GA Type	Eval /Gen	Selection Type	Size (Ratio)	XO	ρ 0.1	0.2	0.5	1.0
SGA	120	wheel		X	72.8	68.9	64.4	62.5
	120	tourn	2	X	87.8	79.1	66.2	58.2
	120	tourn	5	X	93.1	84.3	67.7	55.8
	120	tourn	12	X	94.9	86.5	68.2	54.6
	120	trunc	0.1	X	95.0	86.8	68.2	54.3
	120	trunc	0.1	-	89.8	78.9	63.0	52.6
	12	tourn	2	X	93.6	90.5	81.3	67.0
	12	tourn	5	X	97.6	95.1	86.5	71.2
	12	tourn	12	X	**98.1**	**95.9**	**88.0**	73.1
	12	tourn	2	-	89.1	86.0	76.9	63.7
	12	tourn	5	-	96.1	92.9	83.3	67.7
	12	tourn	12	-	97.4	94.7	85.9	70.4
	12	trunc	0.2	X	**97.7**	**95.4**	87.3	72.3
	12	trunc	0.3	X	97.6	95.1	86.7	71.6
	12	trunc	0.1	-	97.6	95.2	87.1	72.0
	12	trunc	0.3	-	96.6	93.5	83.8	68.1
RIGA	120	wheel		X	73.9	70.8	66.7	64.8
	120	tourn	2	X	87.0	78.4	69.7	67.3
	120	tourn	5	X	92.6	83.6	73.2	72.4
	120	tourn	12	X	94.5	85.8	75.5	75.3
	120	trunc	0.1	X	94.6	86.0	75.6	75.9
	120	trunc	0.1	-	89.5	78.6	67.5	67.1
	12	tourn	2	X	91.8	88.6	82.4	82.0
	12	tourn	5	X	96.9	94.1	87.3	87.0
	12	tourn	12	X	97.6	95.1	**88.5**	**88.3**
	12	tourn	2	-	88.3	85.1	78.1	77.8
	12	tourn	5	-	95.3	92.0	84.0	83.7
	12	tourn	12	-	96.8	93.9	86.4	86.1
	12	trunc	0.2	X	97.1	94.4	87.6	**87.4**
	12	trunc	0.3	X	96.7	93.9	87.0	86.7
	12	trunc	0.1	-	97.1	94.4	87.3	87.1
	12	trunc	0.3	-	95.2	91.9	83.8	83.4

Results from the One-Max problem are presented in tables 2 ($\tau = 10$) and 3 ($\tau = 50$). The most obvious point is that low-intensity selection (small tournaments and roulette wheel selection) performed very poorly relative to the other settings. Given the smoothness of the fitness function, this is not surprising - the task is essentially one of chasing a ball across a billiard table as it tilts first one way, then another - little exploration is required. The best performing treatment always used size 12 tournaments and small populations, whatever the setting. In these small populations, size 12 tournaments will be very close to truncation selection (in a population of 12, a size 12 tournament will fail to generate

[1]The combination of crossover with truncation to the best individual is pointless, and so was not used.

Table 3: Mean BOG Fitness, One-Max Problem, $\tau = 50$

GA Type	Eval /Gen	Selection		XO	ρ			
		Type	Size (Ratio)		0.1	0.2	0.5	1.0
SGA	120	wheel		X	79.3	76.9	71.4	65.4
	120	tourn	2	X	97.3	94.1	83.2	65.3
	120	tourn	5	X	98.8	96.8	88.5	70.4
	120	tourn	12	X	99.1	97.4	90.3	74.1
	120	trunc	0.1	X	99.1	97.5	90.6	74.8
	120	trunc	0.1	-	98.4	95.7	83.8	62.5
	12	tourn	2	X	96.1	95.4	93.4	89.9
	12	tourn	5	X	99.4	98.9	97.2	93.9
	12	tourn	12	X	**99.6**	**99.2**	**97.6**	94.5
	12	tourn	2	-	91.7	91.1	89.0	85.5
	12	tourn	5	-	98.5	97.9	95.8	92.1
	12	tourn	12	-	99.4	98.9	97.1	93.7
	12	trunc	0.2	X	**99.5**	**99.1**	**97.4**	94.2
	12	trunc	0.3	X	**99.5**	99.0	97.2	94.0
	12	trunc	0.1	-	**99.5**	99.0	97.3	94.1
	12	trunc	0.3	-	99.0	98.3	96.3	92.6
RIGA	120	wheel		X	79.1	77.3	74.3	73.4
	120	tourn	2	X	97.0	93.7	87.5	87.1
	120	tourn	5	X	98.7	96.6	92.7	92.8
	120	tourn	12	X	99.0	97.3	94.0	94.1
	120	trunc	0.1	X	99.0	97.4	94.1	94.3
	120	trunc	0.1	-	98.4	95.6	86.9	86.7
	12	tourn	2	X	94.4	93.7	92.3	92.2
	12	tourn	5	X	99.0	98.5	97.0	97.0
	12	tourn	12	X	**99.5**	99.0	**97.6**	**97.6**
	12	tourn	2	-	90.9	90.2	88.6	88.5
	12	tourn	5	-	97.9	97.2	95.5	95.4
	12	tourn	12	-	99.1	98.5	96.9	96.8
	12	trunc	0.2	X	99.2	98.6	97.2	**97.2**
	12	trunc	0.3	X	98.9	98.3	96.9	96.8
	12	trunc	0.1	-	99.2	98.7	97.2	97.1
	12	trunc	0.3	-	97.9	97.2	95.4	95.4

one of the two fittest individuals less than 1% of the time) – yet even this tiny amount of extra stochasticity slightly improved the performance. It also always used crossover, though again the differences between treatments with and without crossover were tiny – but consistent. The other major theme we can see in these results is that for small disruption ($\rho < 0.3$), random immigrants were very slightly deleterious, whereas they were beneficial when $\rho \geq 0.5$, and substantially so for $\rho = 1.0$. Interestingly, while the actual performance differed greatly between rapid and slow change ($\tau = 10$ vs $\tau = 50$), the relative performance of the different algorithm settings varied little.

Table 4: Mean BOG Fitness, Royal Road Problem, $\tau = 10$

GA Type	Eval /Gen	Selection		XO	ρ			
		Type	Size (Ratio)		0.1	0.2	0.5	1.0
SGA	120	wheel		X	44.7	35.9	27.2	38.9
	120	tourn	2	X	55.3	39.2	26.8	47.4
	120	tourn	5	X	69.0	47.4	28.8	**49.3**
	120	tourn	12	X	73.8	50.8	29.9	**49.3**
	120	trunc	0.1	X	73.5	50.9	29.5	48.6
	120	trunc	0.1	-	58.9	39.2	24.3	45.0
	12	tourn	2	X	68.7	59.5	42.5	35.1
	12	tourn	5	X	**81.2**	**68.8**	46.5	36.6
	12	tourn	12	X	**82.8**	**70.1**	47.0	36.6
	12	tourn	2	-	58.3	50.8	37.4	31.9
	12	tourn	5	-	75.0	63.0	42.6	34.3
	12	tourn	12	-	78.7	65.8	43.8	34.8
	12	trunc	0.2	X	79.4	66.2	43.6	34.6
	12	trunc	0.3	X	79.8	66.8	44.2	35.0
	12	trunc	0.1	-	77.6	64.6	42.3	33.9
	12	trunc	0.3	-	75.8	63.0	42.1	33.9
RIGA	120	wheel		X	47.1	37.1	29.1	40.0
	120	tourn	2	X	53.3	39.2	29.1	46.2
	120	tourn	5	X	67.0	46.4	33.5	**48.9**
	120	tourn	12	X	71.5	49.6	36.4	**48.9**
	120	trunc	0.1	X	71.2	49.4	36.3	47.9
	120	trunc	0.1	-	57.8	38.8	29.7	43.6
	12	tourn	2	X	63.6	54.9	44.4	42.9
	12	tourn	5	X	77.9	65.3	**48.7**	46.8
	12	tourn	12	X	79.8	66.6	**48.8**	47.3
	12	tourn	2	-	55.9	48.6	39.4	38.7
	12	tourn	5	-	71.8	59.9	44.6	43.7
	12	tourn	12	-	75.7	62.5	45.7	44.7
	12	trunc	0.2	X	76.5	63.1	45.9	44.2
	12	trunc	0.3	X	76.0	63.4	46.5	44.4
	12	trunc	0.1	-	74.8	61.6	44.6	43.3
	12	trunc	0.3	-	71.4	59.3	43.7	42.7

Table 5: Mean BOG Fitness, Royal Road Problem, $\tau = 50$

GA Type	Eval /Gen	Selection		XO	ρ			
		Type	Size (Ratio)		0.1	0.2	0.5	1.0
SGA	120	wheel		X	63.4	56.0	43.8	40.1
	120	tourn	2	X	86.8	74.0	49.2	44.2
	120	tourn	5	X	94.3	84.8	57.6	45.7
	120	tourn	12	X	**95.3**	86.6	59.0	45.5
	120	trunc	0.1	X	94.4	84.6	57.1	44.6
	120	trunc	0.1	-	90.1	76.2	49.1	42.0
	12	tourn	2	X	78.4	75.9	69.5	61.7
	12	tourn	5	X	94.2	**90.4**	80.2	68.3
	12	tourn	12	X	**95.7**	**91.8**	**81.1**	68.9
	12	tourn	2	-	66.1	64.1	59.4	54.2
	12	tourn	5	-	88.7	85.1	75.4	64.3
	12	tourn	12	-	93.5	89.2	78.0	65.8
	12	trunc	0.2	X	94.1	89.6	77.6	64.5
	12	trunc	0.3	X	93.8	89.7	78.3	65.4
	12	trunc	0.1	-	93.4	88.6	76.2	63.0
	12	trunc	0.3	-	90.8	86.7	75.7	63.6
RIGA	120	wheel		X	72.9	61.8	49.2	47.9
	120	tourn	2	X	85.1	71.9	54.4	52.7
	120	tourn	5	X	93.8	83.5	65.7	65.2
	120	tourn	12	X	94.7	85.2	66.6	66.0
	120	trunc	0.1	X	93.6	82.9	65.0	64.0
	120	trunc	0.1	-	89.1	74.7	53.7	52.7
	12	tourn	2	X	72.5	70.3	66.1	65.8
	12	tourn	5	X	91.5	87.6	79.2	**79.0**
	12	tourn	12	X	94.2	89.8	**80.5**	**80.4**
	12	tourn	2	-	63.5	61.6	58.2	58.1
	12	tourn	5	-	85.3	81.6	74.0	73.9
	12	tourn	12	-	91.0	86.6	77.1	77.1
	12	trunc	0.2	X	91.6	87.2	77.3	77.1
	12	trunc	0.3	X	90.1	86.1	77.1	77.1
	12	trunc	0.1	-	91.2	86.5	75.9	75.8
	12	trunc	0.3	-	85.5	81.7	73.5	73.5

For the Royal Road problem, please see tables 4 ($\tau = 10$) and 5 ($\tau = 50$). Here, the scale of change was much more important, with $\rho = 1.0$ behaving very differently from the other settings for $\tau = 10$. In this case only, large populations were beneficial and random immigrants detrimental, though the results still slightly favoured more stringent selection. For less disruptive or slower change, though, the preceding results carried through: large tournaments (or other forms of stringent selection) were highly beneficial, and crossover was generally mildly beneficial. Random immigrants were mildly detrimental except in the case of large change ($\rho = 1.0$), when they were quite beneficial. Typical examples of the dynamic behaviour are illustrated in figure 1; we see clearly that small population settings recovered much more rapidly after disruption, with larger tournaments dominating after the first few epochs.

The situation with the knapsack problem (tables 6 ($\tau = 10$) and 7 ($\tau = 50$)) was somewhat more complex. For the first time, we saw intermediate-level selection (tournament size 5 rather than 12) out-performing the most extreme stringency in selection, though not by huge margins. Weak selection (roulette wheel or small tournaments) still performed dramatically worse. The best performers always used crossover, though again the differences were always small. Random immigrants were generally deleterious when other settings were well chosen, except for the most disruptive change, though they were beneficial when the other settings were poorly chosen. Figure 2 shows a typical example of the dynamic behaviour; while the differences with population size were much less, we nevertheless see small populations combined with size-5 tournaments performing best overall, with size-12 tournaments still giving respectable results.

6. DISCUSSION

The first and most important point in all these results is how little benefit arose from exploration by large populations. In almost all settings, the best performance resulted from small populations (12) with stringent (tournament 5) or extremely stringent (tournament 12) selection. In the

Table 6: Mean BOG Fitness, Knapsack Problem, $\tau = 10$

GA	Eval	Selection		XO				ρ			
Type	/Gen	Type	Size (Ratio)		0.05	0.1	0.2	0.3	0.4	0.5	1.0
SGA	120	wheel		X	1051.5	1022.2	990.3	972.8	961.4	952.6	920.9
	120	tourn	2	X	1187.6	1132.0	1059.0	1014.5	984.5	962.7	909.1
	120	tourn	5	X	**1239.9**	1185.3	1102.0	1043.2	999.6	966.0	883.2
	120	tourn	12	X	**1237.3**	1187.6	1105.8	1045.2	998.2	961.4	874.0
	120	trunc	0.1	X	1235.4	1186.8	1106.6	1045.0	997.9	960.2	871.3
	120	trunc	0.1	-	1205.8	1139.6	1046.4	986.9	944.0	914.0	863.6
	12	tourn	2	X	1180.8	1164.4	1132.2	1103.3	1077.3	1051.7	949.5
	12	tourn	5	X	1229.3	**1202.8**	**1163.5**	**1131.0**	**1102.2**	1076.8	970.3
	12	tourn	12	X	1223.2	1192.3	1150.0	1116.3	1087.9	1063.9	965.4
	12	tourn	2	-	1158.7	1139.5	1105.9	1076.7	1051.2	1026.5	932.3
	12	tourn	5	-	1219.1	1191.4	1148.1	1114.1	1085.2	1058.9	954.5
	12	tourn	12	-	1219.1	1188.0	1144.7	1110.6	1081.8	1056.9	957.7
	12	trunc	0.2	X	1220.1	1189.7	1147.5	1114.0	1085.7	1061.9	965.1
	12	trunc	0.3	X	1221.2	**1192.8**	**1151.7**	**1118.8**	1091.4	1065.8	966.8
	12	trunc	0.1	-	1217.2	1186.7	1141.3	1108.6	1081.3	1055.3	959.9
	12	trunc	0.3	-	1217.3	1186.2	1142.2	1106.6	1078.4	1052.2	952.1
RIGA	120	wheel		X	1050.7	1028.9	1004.8	989.3	980.1	972.5	958.1
	120	tourn	2	X	1174.9	1122.5	1060.6	1027.1	1006.3	992.1	966.3
	120	tourn	5	X	1233.3	1179.3	1101.8	1059.3	1035.5	1022.9	1011.0
	120	tourn	12	X	1230.4	1181.4	1111.3	1069.3	1049.3	1040.4	1036.8
	120	trunc	0.1	X	1217.5	1172.5	1106.9	1068.2	1049.5	1042.8	1041.0
	120	trunc	0.1	-	1179.2	1122.0	1053.6	1018.0	1002.0	993.8	989.4
	12	tourn	2	X	1144.6	1128.6	1103.0	1085.5	1072.1	1065.7	1059.8
	12	tourn	5	X	1206.3	1180.0	1143.1	1118.5	1101.2	**1092.0**	**1088.0**
	12	tourn	12	X	1192.9	1164.8	1128.4	1103.9	1088.5	1079.7	1076.7
	12	tourn	2	-	1137.8	1118.4	1088.8	1066.8	1053.2	1045.8	1041.5
	12	tourn	5	-	1193.4	1164.4	1127.0	1102.3	1085.8	1076.1	1072.7
	12	tourn	12	-	1187.4	1157.7	1121.1	1097.7	1081.2	1074.0	1070.4
	12	trunc	0.2	X	1191.5	1162.9	1127.1	1103.5	1088.8	1080.0	1077.2
	12	trunc	0.3	X	1201.0	1173.9	1136.7	1112.9	**1095.5**	**1087.4**	**1083.3**
	12	trunc	0.1	-	1178.8	1150.0	1116.8	1094.0	1080.0	1071.3	1068.7
	12	trunc	0.3	-	1190.0	1160.5	1123.1	1097.9	1080.4	1072.0	1068.9

Table 7: Mean BOG Fitness, Knapsack Problem, $\tau = 50$

GA	Eval	Selection		XO				ρ			
Type	/Gen	Type	Size (Ratio)		0.05	0.1	0.2	0.3	0.4	0.5	1.0
SGA	120	wheel		X	1091.4	1079.6	1058.6	1040.6	1024.5	1010.0	947.1
	120	tourn	2	X	1249.5	1230.6	1194.7	1159.8	1126.6	1094.7	957.3
	120	tourn	5	X	**1283.4**	**1267.8**	1238.3	1208.1	1177.7	1147.5	1000.9
	120	tourn	12	X	1279.2	1264.6	1237.8	1212.3	1185.6	1159.0	1028.4
	120	trunc	0.1	X	1278.3	1263.8	1237.5	1212.3	1187.2	1161.5	1034.1
	120	trunc	0.1	-	1269.0	1248.0	1211.0	1176.5	1143.0	1110.4	967.5
	12	tourn	2	X	1195.8	1192.1	1185.4	1178.8	1172.5	1166.2	1136.6
	12	tourn	5	X	1262.1	1254.2	**1241.9**	**1231.9**	**1222.6**	**1214.2**	1177.1
	12	tourn	12	X	1267.2	1255.9	**1239.4**	**1227.0**	**1216.4**	1207.0	1168.6
	12	tourn	2	-	1176.1	1171.7	1164.2	1157.4	1150.6	1144.0	1114.2
	12	tourn	5	-	1255.6	1247.1	1233.6	1222.6	1212.7	1203.6	1164.0
	12	tourn	12	-	1265.2	1253.6	1236.9	1224.1	1213.0	1203.6	1163.7
	12	trunc	0.2	X	1265.0	1253.5	1236.8	1224.2	1213.8	1203.9	1165.4
	12	trunc	0.3	X	1261.7	1251.5	1236.6	1225.1	1215.4	1206.3	1168.1
	12	trunc	0.1	-	1264.7	1252.7	1235.5	1222.6	1211.4	1201.6	1162.6
	12	trunc	0.3	-	1259.6	1249.0	1233.2	1220.4	1209.7	1199.6	1159.0
RIGA	120	wheel		X	1080.5	1071.9	1058.0	1048.0	1040.0	1033.5	1027.0
	120	tourn	2	X	1241.0	1221.1	1185.7	1156.2	1136.9	1126.9	1119.1
	120	tourn	5	X	**1280.5**	**1265.0**	1236.8	1212.2	1196.8	1193.1	1192.7
	120	tourn	12	X	1275.0	1259.7	1235.1	1214.2	1202.0	1199.4	1199.0
	120	trunc	0.1	X	1269.7	1253.4	1230.4	1212.6	1201.6	1199.7	1200.8
	120	trunc	0.1	-	1257.2	1233.1	1199.7	1172.9	1152.9	1144.6	1143.6
	12	tourn	2	X	1160.2	1156.6	1150.8	1146.5	1143.4	1141.6	1140.2
	12	tourn	5	X	1245.5	1237.6	1226.8	1218.1	1211.9	**1208.8**	**1207.5**
	12	tourn	12	X	1251.6	1240.3	1225.3	1215.3	1207.8	1204.1	**1203.2**
	12	tourn	2	-	1158.2	1153.9	1147.1	1141.3	1137.3	1135.3	1134.3
	12	tourn	5	-	1238.1	1229.7	1217.9	1209.0	1202.3	1199.0	1197.8
	12	tourn	12	-	1247.3	1236.4	1222.0	1211.4	1204.1	1200.5	1200.0
	12	trunc	0.2	X	1248.7	1237.2	1222.7	1212.8	1205.8	1202.3	1201.5
	12	trunc	0.3	X	1243.0	1234.2	1222.3	1213.4	1206.8	1203.9	1203.0
	12	trunc	0.1	-	1246.0	1233.7	1219.3	1209.4	1202.4	1198.8	1198.0
	12	trunc	0.3	-	1237.7	1228.2	1215.4	1205.8	1199.2	1195.8	1194.8

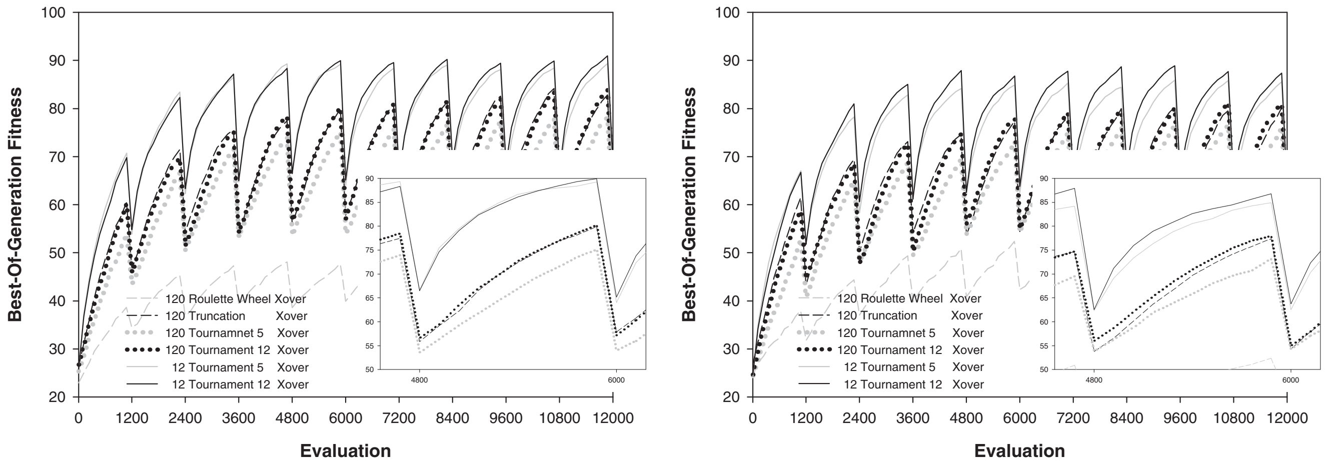

Figure 1: Dynamic Behaviour for Royal Road, $\rho = 0.1$, $\tau = 10$ (left: SGA, right: RIGA)

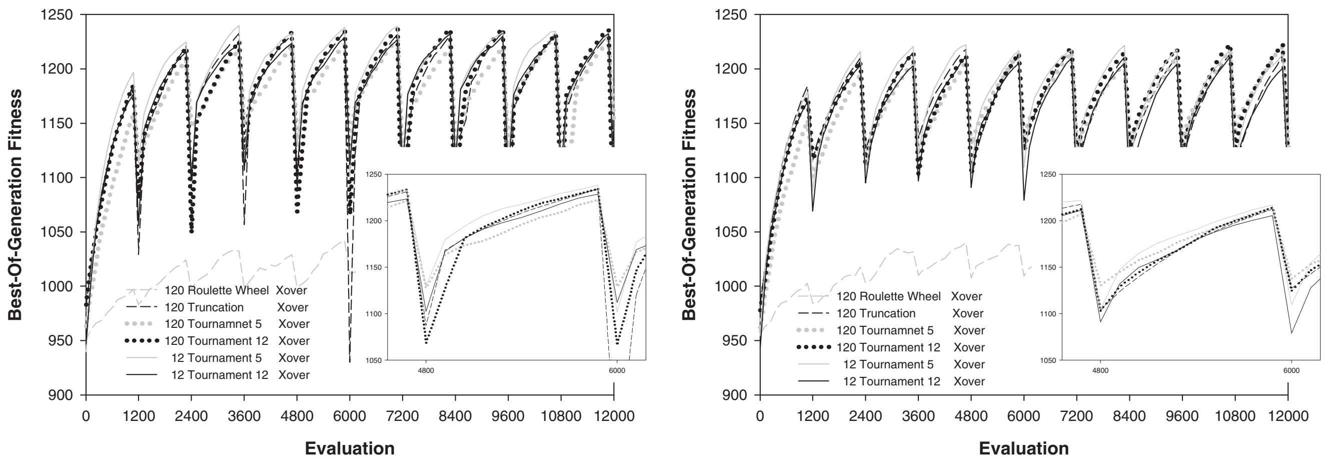

Figure 2: Dynamic Behaviour for Knapsack Problem, $\rho = 0.1$, $\tau = 10$ (left: SGA, right: RIGA)

sole exception, Royal Road with $\rho = 1.0$ and $\tau = 10$, it is worth noting that the change is so extreme and so rapid that it is hard to view it as a dynamic optimisation problem at all – we obtained better performance still by treating it as a series of static problems, restarting the GA after each epoch (Yang's restart GA), with a population of 12 and any tournament size larger than 3.

Even more surprising, in all experiments, the most exploitative setting – population 12, truncation selection to the fittest individual, no crossover – was quite competitive, generally ranking in the top few settings, and always out-performing a 'classical' GA setting of population 120 with roulette wheel selection. This bears out a point made some years ago by Kang [4] in the context of dynamic travelling salesman problems. He noted that there is a common intuition that dynamic optimisation problems are generally tougher than the corresponding static problem – and that it may be wrong. Specifically, the dynamic change in the fitness landscape may permit an exploitative algorithm to escape local optima that would trap it in the corresponding static problem. Preliminary experiments on the tougher problems from the 2009 CEC Dynamic Optimisation Contest [5] have yielded mixed results – small populations yielded better results on the rotated 50-peak and Ackley's functions, but larger populations performed better on the rotated 10-peak and Griewank functions.

Closely related to this is the universally poor performance of roulette wheel selection, which is clearly ill-suited to all of the scenarios examined in these experiments.

Space prevented our inclusion of the detailed EIGA and HIGA results in this paper; it's reasonable to say that they generally followed the same structure as above. EIGA and HIGA did sometimes give the best results overall, but it was always by very small margins over the best settings for SGA and RIGA. We did see more substantial improvements from EIGA and HIGA in other settings – that is, EIGA and HIGA seemed not to lead so much to better performance as to more robust performance, less sensitive to finding exactly the best parameter settings.

Finally, in all problems, crossover was worth including – adding crossover almost universally improved performance. But the improvement was almost always small.

7. CONCLUSIONS

7.1 Assumptions and Limitations

In this work, we assume that the mean best of generation fitness is a suitable metric for comparing the overall performance of evolutionary algorithms. In the absence of any other obvious candidates, we see this as a reasonable assumption.

Our conclusions are preliminary in the sense that they are based on a limited set of problems and only a single form of dynamic change. Clearly, our investigations need to be extended to other problems and other forms of dynamism. However even what has been done to date required many thousands of runs and huge amounts of computer time; such extensions will take time.

7.2 Further Work

The main direction for further work is to investigate how these results extend, both to other underlying problems, and to other forms of dynamism. The earlier results of Kang on continuously dynamic travelling salesman problems suggest that they may extend to some quite different domains.

7.3 Summary

The underlying, and to us surprising, message from these results is that exploitation is not something to be feared in dynamic evolutionary problems. Highly exploitative settings give surprisingly good results, and at least for these problems, a population of 12, combined with tournaments of size 12 and crossover, gave highly competitive results for all problem settings. To answer our original question, 'How hard should we run?', the answer seems generally to be 'Very hard indeed.'

8. ACKNOWLEDGEMENTS

Seoul National University Institute for Computer Technology provided research facilities for this study, which was supported by the Basic Science Research Program of the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (Project No. 2010-0012546), and the BK21-IT program of MEST.

9. REFERENCES

[1] G. Dantzig. Discrete-variable extremum problems. *Operations Research*, pages 266–277, 1957.

[2] J. Grefenstette. Genetic algorithms for changing environments. *Parallel problem solving from nature*, 2:137–144, 1992.

[3] Y. Jin and J. Branke. Evolutionary optimization in uncertain environments – a survey. *IEEE Transactions on Evolutionary Computation*, 9(3):303–317, June 2005.

[4] L. Kang, A. Zhou, B. McKay, Y. Li, and Z. Kang. Benchmarking algorithms for dynamic travelling salesman problems. In *Evolutionary Computation, 2004. CEC2004. Congress on*, volume 2, pages 1286–1292 Vol.2, June 2004.

[5] C. Li, S. Yang, T. T. Nguyen, E. L. Yu, X. Yao, Y. Jin, H. g. Beyer, and P. N. Suganthan. Benchmark generator for cec'2009 competition on dynamic optimization. Technical report, Dept of Computer Science, University of Leicester, 2008.

[6] C. MacNish. Towards unbiased benchmarking of evolutionary and hybrid algorithms for real-valued optimisation. *Connection Science*, 19(4):361–385, 2007.

[7] M. Mitchell, S. Forrest, and J. Holland. The royal road for genetic algorithms: Fitness landscapes and GA performance. In *Towards a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life*, pages 245–254, 1992.

[8] P. Rohlfshagen and X. Yao. Dynamic combinatorial optimisation problems: an analysis of the subset sum problem. *Soft Computing-A Fusion of Foundations, Methodologies and Applications*, pages 1–12, to appear.

[9] J. Schaffer and L. Eshelman. On Crossover as an Evolutionary Viable Strategy. In R. Belew and L. Booker, editors, *Proceedings of the 4th International Conference on Genetic Algorithms*, pages 61–68. Morgan Kaufmann, 1991.

[10] K. Tang, X. Yao, P. Suganthan, C. MacNish, Y. Chen, C. Chen, and Z. Yang. Benchmark functions for the cec'2008 special session and competition on large scale global optimization. Technical report, Nature Inspired Computation and Applications Laboratory, USTC China, 2008.

[11] D. Wolpert, W. Macready, I. Center, and C. San Jose. No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1(1):67–82, 1997.

[12] S. Yang. Non-stationary problem optimization using the primal-dual genetic algorithm. In *Proceedings of the 2003 IEEE Congress on Evolutionary Computation*, volume 3, pages 2246–2253. IEEE Press, 2003.

[13] S. Yang. Genetic algorithms with elitism-based immigrants for changing optimization problems. In *Proceedings of the European Evolutionary Computing Workshops (Evoworkshops)*, volume 4448 of *Springer Lecture Notes in Computer Science*, pages 627–636, Berlin, 2007. Springer-Verlag.

[14] S. Yang and R. Tinos. Hyper-selection in dynamic environments. In *Proceedings of the 2008 IEEE Congress on Evolutionary Computation*, pages 3185–3192. IEEE Press, 2008.

[15] S. Yang and X. Yao. Population-based incremental learning with associative memory for dynamic environments. *IEEE Transactions on Evolutionary Computation*, 12(5):542–561, Oct 2008.