Informative Performance Metrics for Dynamic Optimisation Problems

Stefan Bird
School of Computer Science and IT
RMIT University
Melbourne, Australia
stbird@seatiger.org

Xiaodong Li
School of Computer Science and IT
RMIT University
Melbourne, Australia
xiaodong@cs.rmit.edu.au

ABSTRACT

Existing metrics for dynamic optimisation are designed primarily to rate an algorithm's overall performance. These metrics show whether one algorithm is better than another, but do not indicate any specific aspects of the performance. In this paper we split the offline error metric into two component parts. We propose a new metric to measure convergence speed, and show how this, when combined with a population diversity metric, correlates strongly with the overall performance.

We then use these metrics to analyse several optimisation algorithms, yielding new insight into both the test function and how the algorithms' characteristics can be improved.

Categories and Subject Descriptors

G.1 [**Numerical Analysis**]: Optimisation; F.2.1 [**Analysis of Algorithms and Problem Complexity**]: Numerical Algorithms and Problems

General Terms

Algorithms

Keywords

Evolutionary Computation, Particle Swarms, Multimodal Function Optimisation, Dynamic Optimisation

1. INTRODUCTION

Dynamic optimisation poses a unique challenge for Evolutionary Algorithms. Many algorithms, while being highly effective on static problems, flounder in a changing environment. Diversity preservation becomes critical; not only must the algorithm avoid premature convergence, it must also track "promising" solutions in case they lead to a global optimum in the future.

While there exist many metrics designed to compare performance on dynamic functions, in general they only measure the overall performance of an algorithm without giving insight into its strengths and weaknesses. This information would be very useful to those trying to improve existing algorithms or develop new ones. With this knowledge efforts can be directed more effectively, leading to reduced research time and ultimately, better algorithms.

Algorithms are often framed in terms of exploration vs. exploitation, yet these aspects are rarely actually measured or reported on in papers. This work combines two metrics that exclusively measure the exploration and exploitation characteristics of an algorithm.

We will be presenting a new metric, Best Known Peak Error (BKPE), to measure convergence speed. By combining this with the existing peak cover metric [5], we will offer new insight into the relative strengths of several optimisation algorithms on a dynamic benchmark function. We will also show how and why these metrics correlate with offline error performance [5].

This paper is structured as follows: Section 2 introduces the existing metrics, Moving Peaks and Particle Swarm Optimisation (PSO), the algorithm we will use to demonstrate the metrics. The new metric will be explained in Section 3 and related to offline error in Section 4, followed by the experimental setup in Section 5. The correlation between offline error and the metrics will be discussed in Section 6 as well as how the relative strengths of several PSO variants can be shown. Finally Section 7 will conclude the paper and suggest directions for future research.

2. RELATED WORK

In this paper we introduce a new metric for dynamic optimisation problems, combining it with an existing metric. This section outlines some of the previously developed metrics, as well as the algorithms and dynamic benchmark function we will test the metrics on.

2.1 Performance Metrics

Various metrics have been proposed to measure algorithm performance in dynamic environments. Many report the fittest value at each generation; an algorithm's performance is analysed by graphing these values and comparing with other algorithms at each generation[1, 7].

Online performance [8] measures the average fitness of every individual at each generation. This metric penalises algorithms that keep individuals in suboptimal areas of the decision space to preserve diversity or better react to fitness landscape changes – the suboptimal individuals drag the average population fitness down.

Morrison introduced the Collective Mean Fitness [13]. This

GECCO'07, July 7–11, 2007, London, England, United Kingdom.

works by averaging the fitness of the best individual in each generation over many iterations. By measuring over a large number of generations, this metric provides a stable value with which performance can be compared.

Mean Tracking Error [13] measures the average distance in the search space between the optimum and the best known solution. This indicates the convergence speed of the algorithm on the correct peak. Algorithms which do not track the correct peak are penalised by this metric, even if they have converged on an area of near-optimal fitness.

For the Moving Peaks optimisation benchmark (See Section 2.2), Branke's modified offline error [5] is the most commonly used metric, although it is also used with other fitness functions. This metric measures the average fitness difference between the global optimum and the best found solution since the last landscape change. An algorithm that can converge quickly on a new optimum is rewarded with a low error, as is one that maintains good population diversity to react rapidly to change.

While these metrics are effective in comparing algorithms, they do not say how the algorithms can be improved. The two orthogonal aspects of an algorithm's performance – exploration and exploitation – are measured as one, discarding valuable information.

A less well-known metric outlined by Branke in Chapter 5 of [5] is peak cover. This exclusively measures population diversity – the exploration aspect of an algorithm – by analysing how many of the optima or peaks the algorithm tracks. An algorithm that tracks more peaks is better positioned to adapt once a change occurs; it has a higher chance of already covering the new best peak. For fitness functions where peaks can become submerged or hidden below other parts of the fitness landscape, the hidden peaks are ignored. This prevents the metric penalising an algorithm for not tracking a peak that isn't visible anyway. Peak cover is calculated using Equation (1).

$$peak\ cover = \frac{covered\ peaks}{non\text{-}hidden\ peaks} \tag{1}$$

A peak is covered when there is at least one individual within its catchment area. Figure 1 shows a number of individuals on different peaks. Peak 1 is ignored since its tip is hidden by Peak 2. Peak 3 is not covered as there are no individuals within its catchment area. This gives a peak cover of 0.5 (50%).

In Section 3 we will be introducing the compliment to peak cover, a metric that measures the exploitation of an algorithm. Together, these metrics provide a far more detailed understanding of an algorithm's performance characteristics.

2.2 Moving Peaks

Moving Peaks is a dynamic optimisation problem generator proposed by Branke [4], in which there are a number of peaks that can be specified to move randomly. The object is to track the location of the highest peak as it moves through the decision space.

After a predefined number of evaluations, the peak locations, heights and widths are changed. The currently highest peak may not always be; it may be overtaken by another peak or even hidden completely (Figure 2). To achieve good results, algorithms must track as many peaks as possible.

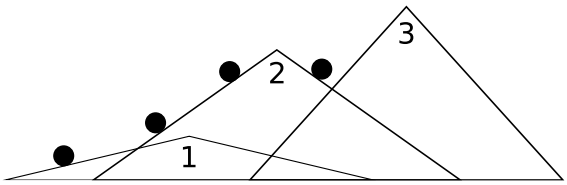

Figure 1: Half of the non-hidden peaks are represented by the population, giving a peak cover of 0.5 (50%). Peak 1 is ignored because it's hidden by Peak 2.

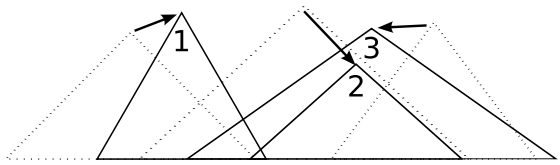

Figure 2: The Moving Peaks function provides a challenge to evolutionary algorithms. The previously-best Peak 2 is now hidden by Peak 3 and Peak 1 has become the new global optimum. Any algorithm that fully converges on a single peak will perform poorly on this function.

2.3 Particle Swarms

Particle Swarm Optimisation (PSO) has proved very effective at solving the Moving Peaks function [12]. It consists of a population of particles that move around the decision space [9]. Each particle has a memory of the best location it has so far visited, known as the personal best, and is able to interact with a number of other particles as defined by the neighbourhood topology [10]. To decide where to move next, the particle randomly chooses a point near the best point it has found and the best point that any of its neighbours has found, known as the neighbourhood best. To prevent particles from travelling directly to the chosen point, they have an inertia. While they are attracted to the chosen point, it may take several iterations to turn around and reach it. The constriction coefficient [6, 9] variation of PSO has been adopted for our experiments, as shown in Equations (2) and (3).

$$\vec{v}_{(i,t+1)} = \chi(\vec{v}_{(i,t)} + \varphi_1(\vec{p}_{(i,t)} - \vec{x}_{(i,t)}) + \varphi_2(\vec{p}_{(g,t)} - \vec{x}_{(i,t)})) \tag{2}$$

$$\vec{x}_{(i,t+1)} = \vec{x}_{(i,t)} + \vec{v}_{(i,t+1)}, \tag{3}$$

where:

$$\varphi_1 = c_1 r_1, \quad \varphi_2 = c_2 r_2,$$

$$\chi = \frac{2\kappa}{\left|2 - c - \sqrt{c^2 - 4c}\right|}$$

t represents the current time step. $\vec{x}_{(i,t)}$ and $\vec{v}_{(i,t)}$ are the current location and velocity of the particle respectively. $\vec{p}_{(i,t)}$ is the personal best location of the current particle and $\vec{p}_{(g,t)}$ the best location found by any of its neighbours. c_1 and c_2 are constants, typically 2.05. $c = c_1 + c_2$. κ is also a constant, usually set at 1. r_1 and r_2 are uniform random numbers between 0 and 1.

χ acts to constrict the velocity of the particle. This prevents it from violently oscillating around its personal best [6] and ensures the surrounding area will be thoroughly explored. In addition, the velocity is limited to the size of the decision space. If a particle's velocity would take it out of the decision space, it is "bounced" off the edge.

For the standard PSO, we have used a von Neumann neighbourhood topology. This was shown in [10] to have good convergence properties while remaining robust against premature convergence. While it cannot compete with algorithms that have been specifically designed for multimodal environments, it provides reasonable performance without incurring heavy code or runtime overheads.

2.4 Niching Particle Swarms

Niching particle swarms offer improved performance on multimodal problems by identifying "niches" within the population and preventing particles from different niches communicating. Each niche maintains a subset of the population on a different peak, allowing the system to converge on multiple optima simultaneously. Even on unimodal problems this can be an advantage, as the added diversity reduces the risk of premature convergence; even if one niche gets trapped in a local optimum, the rest of the population is free to explore other areas. We will be testing the metrics using three niching (also known as specied) particle swarms: SPSO [11], ESPSO [3] and ANPSO [2].

SPSO introduces the concept of species seeds [11]. An individual is a species seed if there are no fitter individuals within a user-specified distance r of it. Particles that are not species seeds are allocated to the fittest seed within r of their location. Each species is defined as the seed and all of the particles that have been allocated to it. Particles within the same species are able to interact freely with each other but are prevented from interacting with any particles from other species.

One of the disadvantages of SPSO is that it is sensitive to the value of r - if it is too small, the level of particle interaction is too low and the population is likely to become trapped in local optima. If it is set too large the algorithm is unable to differentiate nearby peaks - it is very difficult for the system to maintain individuals on multiple peaks within the same species. ESPSO [3] was developed to counter this difficulty. Once a species seed has remained in the same place for several iterations, its members can no longer join or leave. This allows new species to form on nearby optima without interference from optima that have already been discovered. This characteristic makes the algorithm far less sensitive to r - the user can set it to a large arbitrary value with very little impact on performance.

The final niching algorithm we will use is ANPSO [2]. This algorithm analyses which particles tend to stay together over successive iterations and groups those into niches. The rest of the population is placed in a von Neumann neighbourhood topology to increase the speed at which new niches are found, while allowing different promising areas to be explored simultaneously.

2.5 Guaranteed Convergence PSO

A particle swarm can converge quickly on an optimum when there are a number of particles in the area. When there are only a few particles, the system takes a lot longer to accurately find the optimum. GCPSO [15] can be implemented on top of a normal PSO in order to increase convergence speed. It modifies the behaviour so that the fittest particle randomly tries points near its personal best, rather than moving around as the rest of the population does. The radius of the hypersphere in which it searches is adaptively determined by the number of consecutive moves that led to success or failure. A successful move is one that gives the particle a better fitness than it did before, and the opposite for a failure. If the number of successful moves reaches a threshold the radius is increased, making the particle more aggressive in its searches. Conversely, if the number of failures in a row reaches a threshold, the radius is decreased.

The GCPSO algorithm was modified so it could be used with non fully-connected neighbourhood topologies [14]. In this variant, any particle who had a higher personal best fitness than any of its neighbours used the GCPSO searching algorithm and maintained its own success and failure counts. This modification allows it to be used with niching algorithms such as the ones outlined above.

3. MEASURING CONVERGENCE SPEED

Offline error is an effective way to compare an algorithm's overall performance on Moving Peaks. In this section, we introduce a new metric called "Best Known Peak Error" (BKPE), and show how it and peak cover represent the component parts of offline error. Unlike offline error, these metrics are calculated at the end of every generation instead of at each evaluation (See Figure 3).

BKPE measures the convergence speed of the algorithm once it has found the catchment area of a peak. The peak used for the calculation is the peak with the fittest individual that was covered for the entire time since the last peak movement. This metric is calculated similarly to offline error; it is the difference in fitness between the top of the peak and the fittest individual within the peak's catchment area. While the actual peak chosen has little effect on the metric, we chose the best known peak as it's the one the algorithm is most likely to be concentrating its resources on. As with peak cover, individuals on hidden peaks are ignored.

At the end of each generation, for each peak the fittest individual within that peak's catchment area is chosen, and the fitness difference between the individual and the peak is added to that peak's error (See Equation (4) and Figure 3). Immediately before the peaks are moved, the peak with the fittest individual and that was covered at the end of every generation since the last move is selected, and its error added to the total. After this, the error for each peak is reset to 0. The BKPE is calculated by dividing the total error by the number of generations[1]. This is described in detail in Algorithm 1.

$$BKPE = \frac{1}{G}\sum_{g=1}^{G} e_{g,i} \tag{4}$$

[1]If there was no peak covered for every generation between two peak movements, the BKPE cannot be calculated for this period and the generations between the movements are ignored.

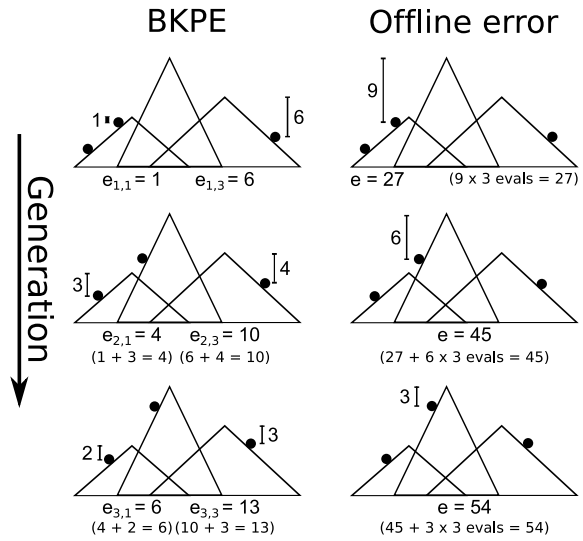

Figure 3: For the BKPE the error is calculated to every known peak. Immediately before a peak movement, the error of the fittest particle on a known peak (in this case the right-hand peak) is added to the total error for the run. The errors for each peak are then reset to 0. The BKPE in the scenario above would be $\frac{13}{3\ generations} = 4.33$. The middle peak is ignored because it was unknown during the first generation. The offline error would be $\frac{54}{9\ evaluations} = 6$.

G is the total number of generations, g is the current generation number and i is the index of the peak that has the fittest individual. $e_{g,i}$ represents the accumulated fitness error between the last peak movement and generation g for peak i.

4. RELATING PEAK COVER AND BKPE TO OFFLINE ERROR

Peak cover and BKPE are orthogonal component parts of offline error. To obtain a good offline error score, the algorithm must converge quickly (measured by BKPE) and track as many peaks as possible (measured by peak cover). An algorithm with a low BKPE and high peak cover will achieve a low offline error, as shown in Figure 4. Conversely, an algorithm that has a high BKPE and low peak cover is likely to have a high offline error.

Let us imagine an algorithm possessing an oracle. The oracle knows the exact location of each peak but not the height. It must keep an individual on each peak in order to return the global optimum. This algorithm would have a peak cover of 1 and a BKPE of very close to 0. Since we track every peak, the best known peak will always be the globally optimal peak, thus the BKPE and offline error will be almost[2] the same.

[2]There will be a small difference – the offline error is computed every evaluation whereas the BKPE is calculated every generation.

r is the cumulative total error, initially 0
m is the number of ticks for which the metric has been computed, initially 0
$BKPE = \frac{r}{m}$
t is the current tick
l is the tick of the last peak movement
K is the set of peaks
U is the set of unknown peaks
P is the set of population members
e_y is the error of the peak y
$covering(\vec{k_i})$ returns the subset of P that covers $\vec{k_i}$
$hidden()$ returns the subset of K that is hidden by other peaks
$fitness(\vec{x})$ returns the fitness of point $\vec{x}$
y and f are temporary variables

At the end of each generation:

for $\forall i; \vec{k_i} \in K \wedge \vec{k_i} \notin U$ **do**
  $C \leftarrow covering(\vec{k_i})$
  if $C = \oslash$ **then**
    $U \leftarrow U \cup \vec{k_i}$
  end
  else
    // Find the fittest individual on peak $\vec{k_i}$
    $y \leftarrow 0$
    for $\forall j; \vec{c_j} \in C$ **do**
      if $fitness(\vec{c_j}) > fitness(\vec{c_y})$ **then**
        $y \leftarrow j$
      end
    end
    $e_i \leftarrow e_i + fitness(\vec{k_i}) - fitness(\vec{c_y})$
  end
end

Immediately before a peak movement:

if $\exists i; \vec{k_i} \in K \wedge \vec{k_i} \notin U$ **then**
  // Find the fittest individual on a non-covered
  // peak and note which peak it's on
  $f \leftarrow -\infty$
  $y \leftarrow 0$
  for $\forall i; \vec{k_i} \in K \wedge \vec{k_i} \notin U$ **do**
    $C \leftarrow covering(\vec{k_i})$
    for $\forall j; \vec{c_j} \in C$ **do**
      if $fitness(\vec{c_j}) > f$ **then**
        $y \leftarrow i$
        $f \leftarrow fitness(\vec{c_j})$
      end
    end
  end
  $r \leftarrow r + e_y$
  $m \leftarrow m + t - l$
end

Immediately after a peak movement:

$U \leftarrow hidden()$
$l \leftarrow t$
for $\forall i; \vec{k_i} \in K \wedge \vec{k_i} \notin U$ **do**
  $e_i \leftarrow 0$
end

Algorithm 1: The procedure for calculating BKPE.

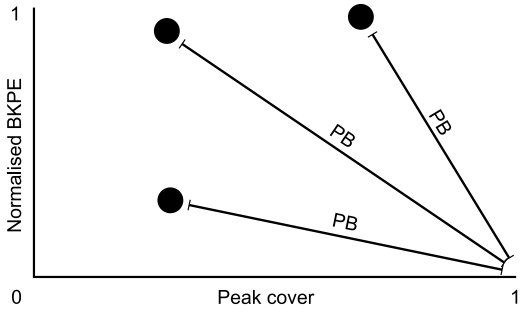

Figure 4: Algorithms that get a low BKPE and high peak cover will have a low offline error, thus offline error is related to the distance from the bottom right of the graph.

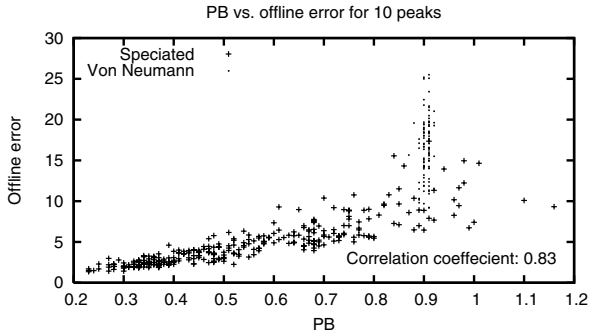

Figure 5: The correlation between PB and offline error. The vertical bar represents the runs using the von Neumann topology; these all had a small BKPE and a peak cover of 0.1. Across all the runs, the correlation coefficient between offline error and PB was 0.83, showing these are strongly correlated.

5. EXPERIMENTAL SETUP

We are primarily interested in showing the correlation of peak cover and BKPE with offline error. To do that, we will be comparing the offline error of each run to the distance from the point as it would be plotted on Figure 4 to the bottom right of the graph. The BKPE is normalised so that all values are in the range [0, 1]. The actual calculation for this is shown in Equation (5). We have called the resulting variable PB, standing for "**P**eak cover and **B**KPE".

$$PB = \sqrt{(1 - peak\ cover)^2 + \left(\frac{BKPE}{max(BKPE)}\right)^2} \qquad (5)$$

In addition, we will also analyse the relative performance of four PSO variants and the effect of adding a convergence enhancer to the algorithms. The PSO models we will use are standard constriction PSO with a von Neumann neighbourhood model, SPSO, ESPSO and ANPSO. Each PSO is also tested in combination with GCPSO to enhance convergence. All parameters have been set to the recommended values. An r value of 30 has been used for SPSO and ESPSO. 50 runs with an initial population size of 100 was used for all PSO models and each run was stopped after 500000 evaluations. For SPSO, we limited each species to 10 particles - any excess particles were randomly distributed in the decision space. These parameter settings are the same as were used in [12]. To allow better comparison with ESPSO and encourage more stable species, we used the particle's personal best as the species seed.

The Moving Peaks Scenario 2 benchmark was used, with different numbers of peaks from 1 to 190. The peaks change height and location every 5000 evaluations. To detect these changes, the personal best of the fittest particle in each of the top 5 species is recorded. If the fitness of that point changes, all of the particles have their personal best memory reset to their current location. As the von Neumann topology doesn't maintain distinct species, we monitor the landscape using the 5 fittest particles instead.

6. RESULTS

This section will first discuss the correlation of the metrics with offline error and how it changes with the number of peaks. It will then analyse the differences in the performance of the tested PSO variants. Finally we will use the metrics to determine how the algorithms' performance can be improved.

6.1 Correlation with offline error

Figure 5 shows the Pearson correlation between PB and offline error when there are 10 peaks. The vertical bar at approximately 0.9 on the PB axis is caused by the von Neumann runs. Invariably this topology caused the PSO to converge on only one of the 10 peaks, giving a peak cover of 0.1. As the BKPE was small relative to the other algorithms, almost all of the von Neumann runs had a PB of around 0.9. In this case, the offline error was almost entirely determined by movements of the peak the algorithm converged on – it had very little chance to locate alternate peaks once it had converged. In our tests there was almost no correlation between offline error and PB for the von Neumann runs: -0.01 in the case of the 10-peak tests. To avoid skewing our results, we have excluded these runs when discussing overall correlations.

Figure 6 shows the correlation between the two metrics and offline error for the speciated algorithms as the number of peaks varies. A high correlation indicates that the given metric is a large determining factor in the offline error. To make the graph easier to read, we have plotted the correlation of $1 - peak\ cover$ and made the peaks axis logarithmic. A coefficient of -1 or 1 indicates perfect correlation and 0 indicates none. A negative correlation indicates that as the first variable increases the second generally decreases.

The correlation of PB with offline error is generally slightly higher than both peak cover and BKPE. When there are fewer than 50 peaks the correlation is very strong (> 0.8), showing that these metrics are the main factors of offline error. Above 50 peaks the correlation declines; it becomes increasingly important to track the most promising peaks instead of every peak possible.

The peak cover is most important when there are only a few peaks. Since the number of peaks relative to the pop-

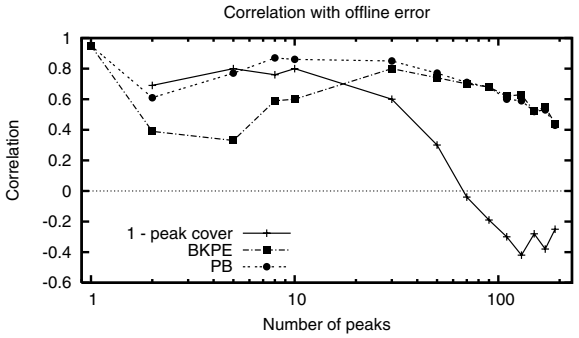

Figure 6: The correlation of each metric depends on the number of peaks.

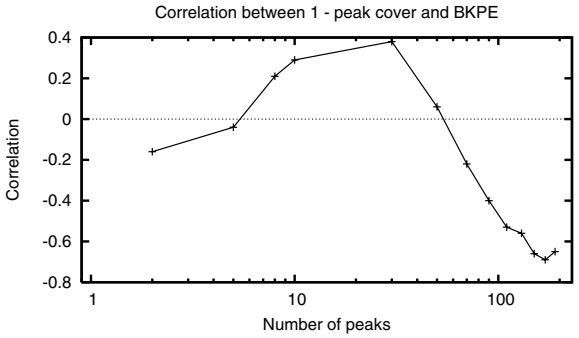

Figure 7: The correlation between 1 - peak cover and BKPE. When there are many peaks, maintaining a high peak cover results in a low convergence speed because there are fewer individuals on each peak.

ulation size is low, it is possible to keep a significant number of individuals on each peak. This allows the algorithm to respond more quickly to peak movements and changes. The comparatively low BKPE correlation shows that while convergence speed does play a role here, the offline error is mainly determined by how many of the peaks were tracked.

As the number of peaks increases, tracking too many becomes a disadvantage. Since the algorithms only have a finite number of individuals available, there is a risk of spreading the search power too thin. Figure 7 shows that as the number of peaks increases, runs where the peak cover was high tended to also have a high BKPE. This indicates that maintaining individuals on too many peaks reduces the algorithm's ability to quickly converge, offsetting any diversity advantage. The correlation between offline error and BKPE indicates quick convergence is important, even if it is not on the correct peak. Although the algorithm may not be tracking the currently-optimal peak, it is likely to be tracking another with similar fitness. As long as the algorithm accurately knows the location of the alternative peaks, the error penalty should be relatively small.

6.2 Algorithm performance

Figures 8 and 9 show the performance of the algorithms on the 10 and 110 peak problems respectively. The most obvious difference between the two is that the peak cover is

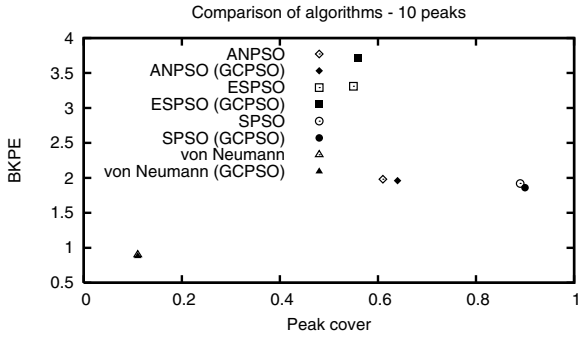

Figure 8: Comparative performance with 10 peaks. Bottom left: The non GCPSO von Neumann point is obscured by the GCPSO-enhanced result as the BKPE and peak cover are almost identical.

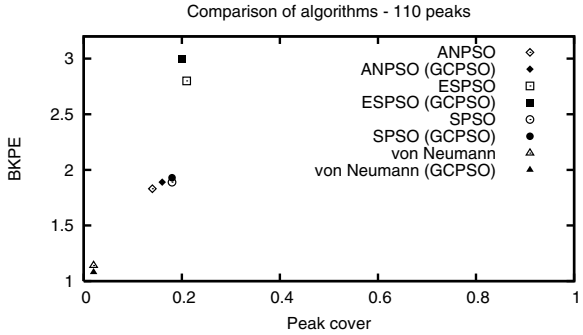

Figure 9: Comparative performance with 110 peaks.

much lower with 110 peaks; the algorithms no longer have enough individuals to represent all of the peaks – they must choose the most promising ones.

To allow easy comparison, the symbol used for the GCPSO-enhanced algorithms is a filled version of the respective unenhanced version. As can be seen in both figures, GCPSO made very little difference to performance. It would seem that on this problem at least, PSO does not require this enhancement to achieve fast convergence.

The number of peaks did not change the algorithms' relative BKPE and peak cover rankings. Von Neumann has an extremely low BKPE because almost all individuals converged on the same peak. The extra search power gives it a far lower BKPE than the niching algorithms, at the cost of only being able to track a single peak and thus a very low peak cover. It should be repeated that von Neumann's offline error performance was very poor as it was difficult for it to switch peaks.

ANPSO and SPSO showed very similar BKPE, although for the 10 peak problem SPSO had a significantly better peak cover. For the 110 peak problem both the BKPE and peak cover are almost identical, resulting in very similar offline errors of 3.69 and 4.04 respectively.

ESPSO's comparatively poor BKPE and peak coverage can be expected as in general it is slower to locate optima than SPSO [3]. It was designed to reduce the sensitivity to SPSO's radius parameter and thus require less tuning. Since the optimal radius value is already known and we are only

Table 1: Algorithm performance with 10 peaks (mean and standard error).

Algorithm	Peak cover	BKPE	Offline error
ANPSO	0.61 (±0.01)	1.98 (±0.07)	4.69 (±0.23)
ANPSO (GCPSO)	0.64 (±0.01)	1.96 (±0.06)	4.04 (±0.16)
ESPSO	0.55 (±0.02)	3.31 (±0.10)	7.43 (±0.35)
ESPSO (GCPSO)	0.56 (±0.03)	3.71 (±0.13)	7.57 (±0.43)
SPSO	0.89 (±0.01)	1.92 (±0.05)	2.63 (±0.08)
SPSO (GCPSO)	0.90 (±0.01)	1.86 (±0.05)	2.24 (±0.08)
von Neumann	0.11 (±0.00)	0.90 (±0.04)	16.39 (±0.57)
von Neumann (GCPSO)	0.11 (±0.00)	0.88 (±0.04)	16.53 (±0.51)

Table 2: Correlation with offline error for 10 peaks.

	ANPSO	ESPSO	SPSO	VN[a]
1 - Peak cover	0.5	0.76	0.12	0.17
BKPE	0.37	-0.04	0.58	0

[a]von Neumann

interested in tracking the highest peak within each species area, the enhancements in ESPSO do not translate to improved performance as measured by these metrics.

Finally, in Table 1 we compare the offline error of the algorithms when tested on the 10 peak problem. SPSO combined with GCPSO was by far the best performing algorithm, while von Neumann was the worst. The correlation between offline error and the presented metrics is evidenced by these results.

6.3 Improving the algorithms

Each algorithm has its own strengths and weaknesses. Some algorithms such as SPSO are very good at maintaining diversity, others are able to converge very quickly. The overall effectiveness of an algorithm is still measured by offline error; as discussed above increasing peak coverage or decreasing BKPE may not necessarily be advantageous.

To find how an arbitrary algorithm can be improved, we determine the correlation of peak cover and BKPE with offline error. Table 2 lists these correlations for the tested algorithms. A high correlation between one of the metrics and offline error indicates the algorithm's final performance is dependant on that aspect. A low correlation suggests improving the algorithm in this area is not likely to impact on overall performance – the effort is better spent elsewhere.

It should be noted that the numbers in Table 2 do not indicate performance. Algorithms that have similar correlation for a particular metric may have wildly different performance in that area. For example, ESPSO and von Neumann both have almost no correlation between BKPE and offline error, however ESPSO's mean BKPE is more than 3 times as large as von Neumann's.

The most striking feature of the table is the very low correlations for the von Neumann runs. Since both metrics have a low correlation, we know the offline error must be determined by some other factor. In this case it is the movements and average height of the algorithm's chosen peak.

ANPSO showed a reasonable correlation for both metrics, indicating that improving either the diversity or convergence speed should improve offline error. Interestingly, SPSO and ESPSO were opposite each other in this regard: for SPSO increasing the peak coverage has a negligible effect, most likely because peak cover was already very high. ESPSO on the other hand benefited greatly from increased peak coverage. ANPSO has a similar peak cover but far less correlation with offline error, possibly indicating that ESPSO is becoming trapped on low value peaks. In any case, the high correlation shows that adding a diversity preservation mechanism to this algorithm would likely improve offline error.

The lack of correlation between BKPE and offline error for ESPSO shows that while the BKPE is very high, improving it is not likely to have much impact on the offline error score. By contrast SPSO had a strong correlation for BKPE - increasing convergence speed should improve the overall performance.

7. CONCLUSION

The metrics presented in this paper allow better insight into algorithm performance than offline error alone. While offline error is an effective metric by which to rate algorithms, it can still be broken down into two principal components: convergence speed and diversity. By measuring the performance of these aspects individually, peak cover and BKPE can better show the strengths and weaknesses of an algorithm.

There are many future research directions from this paper. First of all would be to analyse the performance of other algorithms and improve their diversity and convergence characteristics. Other diversity preservation techniques may be more effective on this function than niching, and techniques such as quantum swarms as presented in [12] may reduce the BKPE better than GCPSO.

It may also be possible to modify both the peak cover and BKPE metrics to better rate the diversity and convergence properties of an algorithm. By weighting each peak by the number of individuals on it, we can analyse how an algorithm is spending its resources. This may encourage the development of algorithms that dynamically allocate individuals to the most promising peaks. By maintaining only a minimal population on suboptimal peaks, an algorithm can increase its convergence speed on good peaks while still being able to react quickly to a low peak becoming the global optimum.

8. REFERENCES

[1] T. Bäck. On the behavior of evolutionary algorithms in dynamic fitness landscapes. In *Proceedings of the 1998 IEEE Congress of Evolutionary Computation (CEC98)*, 1998.

[2] S. Bird and X. Li. Adaptively choosing niching parameters in a PSO. In *GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 3–10, New York, NY, USA, 2006. ACM Press.

[3] S. Bird and X. Li. Enhancing the robustness of a spoliation-based PSO. In *Proceedings of the IEEE International Conference on Evolutionary Computation*, pages 843–850, 2006.

[4] J. Branke. Memory enhanced evolutionary algorithms for changing optimization problems. In Peter J. Angeline, Zbyszek Michalewicz, Marc Schoenauer, Xin Yao, and Ali Zalzala, editors, *Proceedings of the Congress on Evolutionary Computation*, volume 3, pages 1875–1882, Mayflower Hotel, Washington D.C., USA, 6-9 1999. IEEE Press.

[5] J. Branke. *Evolutionary Optimization in Dynamic Environments*. Kluwer Academic Publishers, Norwell, MA, USA, 2002.

[6] M. Clerc and J. Kennedy. The particle swarm - explosion, stability, and convergence in a multidimensional complex space. In *IEEE Transactions on Evolutionary Computation*, volume 2, pages 58–73, 2002.

[7] J. Grenfenstette. Evolvability in dynamic fitness landscapes: A genetic algorithm approach. In *Proceedings of the 1999 IEEE Congress of Evolutionary Computation (CEC99)*, 1999.

[8] K. De Jong. *An analysis of the behavior of a class of genetic adaptive systems.* PhD thesis, University of Michigan, 1975.

[9] J. Kennedy and R. Eberhart. *Swarm intelligence*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2001.

[10] J. Kennedy and R. Mendes. Population structure and particle swarm performance. In *Proceedings of the 2002 Congress on Evolutionary Computation (CEC)*, pages 1671–1676, 2002.

[11] X. Li. Adaptively choosing neighbourhood bests using species in a particle swarm optimizer for multimodal function optimization. In *Proceedings of Genetic and Evolutionary Computation Conference 2004 (GECCO'04) (LNCS 3102)*, pages 105–116, 2004.

[12] X. Li, J. Branke, and T. Blackwell. Particle swarm with spoliation and adaptation in a dynamic environment. In *GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 51–58, New York, NY, USA, 2006. ACM Press.

[13] R. Morrison. *Designing Evolutionary Algorithms for Dynamic Environments*. Springer-Verlag, Berlin, Germany, 2004.

[14] E. Peer, F. van den Bergh, and A. Engelbrecht. Using neighbourhoods with the guaranteed convergence PSO. In *Proceedings of the 2003 IEEE Swarm Intelligence Symposium (SIS '03)*, 2003.

[15] F. van den Bergh and A. Engelbrecht. A new locally convergent particle swarm optimiser. In *Proceedings of the 2002 IEEE International Conference on Systems, Man and Cybernetics*, 2002.