Hooke-Jeeves Revisited

I Moser, Member, IEEE

Abstract—The Hooke-Jeeves (HJ) Pattern Search, which seems to be the most popular choice among the local search algorithms, was used as an alternative to the dimensional local search (DLS), which has provided excellent results in previous work. In this paper, the question whether the well-known Hooke-Jeeves pattern search could outperform the DLS algorithm that was devised somewhat ad-hoc, is to be investigated. The Moving Peaks (MP) function is used as a benchmark. In our experiments, the algorithms performed almost identically well on the problem instances used. However, it was observed that the pattern move, an intrinsic part of the HJ algorithm, hardly contributed to the quality of the outcome, in fact less than the number sequence used as step sizes for both local searches. We provide some investigations into why the pattern move is less successful than most authors— including the original inventors of the Hooke-Jeeves search — seem to anticipate.

I. INTRODUCTION

In previous work [14] a new algorithm for dynamic function optimisation was introduced. The Multi-phase Multi-individual Extremal Optimisation algorithm (MMEO) is a single-individual optimisation approach which consists of several components and has produced excellent results on the Moving Peaks benchmark made available in [7]. A large proportion of the quality of these hitherto unsurpassed results are evidently produced by the local search algorithm component, which is simple and deterministic. In the implementation used to present the record results [14], the local search phase still contained many redundant steps which caused unnecessary function evaluations. Also, the step lengths used for the technique had been chosen intuitively without further investigation. In this work, the development of suitable step lengths for local search is explored, and the local search technique is compared to the well-known local search algorithm proposed by Hooke and Jeeves in 1961 [10], which is still being used successfully for optimising separable optimisation functions. In this paper, we investigate the potential of the well-known Hooke-Jeeves pattern search with the help of the Moving Peaks (MP) function and its standard result quality measure, the offline error.

II. THE MOVING PEAKS BENCHMARK FUNCTION

The Moving Peaks benchmark function [7] provides a dynamic optimisation problem for researchers to test the capabilities of function optimisation algorithms in a dynamic environment. MP consists of a multidimensional, multimodal landscape with moving optima. Formally, the MP function is defined as follows:

\[ F(\hat{x}, t) = \max(B(\hat{x}), \max_{1,m} P(\hat{x}, h_i(t), w_j(t), \overline{p}_i(t))) \]  

where \( B(\hat{x}) \) is the base landscape on which the \( m \) peaks move, with each peak \( P \) having its height \( h \), width \( w \) and location \( \overline{p} \). The location, size and movement of these optima are definable parameters, hence the need for standardising “scenarios” which can be used for algorithm performance measurement. In the literature, benchmark results have been published predominantly for Scenario 2 with 10 optima that move at an average distance of 1/10 of the search space in a random direction.

A. Offline Error

This performance measure is the standard metric researchers use to benchmark algorithm performance on the Moving Peaks function. At every function evaluation, the best known fitness so far is subtracted from the globally optimal fitness. If this fitness is better than all previous submissions, it is added and averaged over all function evaluations. If it is worse, the last best fitness will be added and averaged.

Because of the averaging effect, the metric has a certain bias which rewards the early submission of a good fitness, even if this fitness stems from a location at a distance from the global optimum. It is therefore a good metric for comparing local search algorithms.

The presence of numerous peaks leads to overlaps in slopes and gradually eradicates the “deep valleys” with “really bad” fitnesses, so that results measured by the offline error are not comparable across instances with different numbers of peaks. Therefore, our experiments do not vary the number of peaks in the scenario.

B. Recent Applications

A comprehensive review of optimisation techniques tested on the Moving Peaks function and published by 2006 has been made available on the Moving Peaks web site [7].

Before this study, a PSO-based approach by Blackwell and Branke [3] reported the best offline error of 1.72 from solving Scenario 2 with 10 peaks, closely followed by the Differential Evolution algorithm presented by Mendes and Mohais [12] with an offline error of 1.75 obtained from solving the same settings.

Several authors have applied new algorithms to the benchmark since. Trojanowski [16] developed a “B-Cell Algorithm”, in which \( x \) cells are cloned \( k \) times. One of the \( k \) clones per cell is randomized, the others are mutated. The
best clone replaces the original cell if it has a better fitness.

Trojanowski presented offline errors for this algorithm on Scenario 2 with 2 – 50 peaks. Although he proposed to omit the high initial values typical for an MP trial, the results cannot challenge the scores mentioned above.

A large part of Trojanowski’s contribution [16] compares the results obtained from solving instances with different numbers of peaks. As has been observed before ([5], [8]), increased numbers of peaks will contribute to the overall elevation of the landscape, in which “bad” errors cease to exist. Trojanowski makes the same observation having provided an initial error of 49.79 for Scenario 2 with 48 peaks, while stating values between 164 and 180 for the algorithm solving instances with 2, 8 and 20 peaks.

Trojanowski argues that in order to avoid skewing the offline error, the initial fitness values recorded during the first 10 changes of landscape should be excluded. While the very first fitness values are indeed necessarily obtained from a randomly chosen location, one could argue that the inclusion of these values reflects how fast the algorithm finds its first optima. The faster this happens, the lower the overall offline error will be for a predefined number of function evaluations.

Lung and Dumitrescu [11] present results which surpass the previous best outcome published by Blackwell and Branke. Their offline error on Scenario 2 with 10 peaks is 1.38. The algorithm used is a hybrid of Particle Swarm Optimisation (PSO) and Crowding Differential Evolution (Crowding DE), in which equal populations of both collaborate. The Crowding DE maintains diversity by replacing the closest individual if it is fitter. The PSO’s task is to converge to the global optimum. Whenever a change is detected, the PSO swarm is reinitialized to the Crowding DE population.

The authors also present the offline errors for both components of the algorithm separately. Crowding DE, although on its own far from the quality of the previously published record results, outperforms PSO by a large margin. The authors conclude that the success of their algorithm is caused by the collaboration between the PSO and DE populations.

Wang, Wang and Yang [18] introduce Branke’s [6] technique of employing 3 populations, originally used with an Evolutionary Algorithm (EA) into PSO. The memory-based reinitialisation of the population specialising in exploitation is triggered by the discovery of a peak. The intended diversity maintenance is thus extended to respond not only to change but the challenge of a multi-modal landscape where several optima have to be discovered. Variations of this scheme are explored with the help of graphs plotting the offline error development on Scenario 1, but no numerical results are presented.

Castrogiovanni, Nicosia and Rascunà [9] use the MP function to test the effect of the technique of aging the individuals of various population-based algorithms, such as a Genetic Algorithm, a DE approach and two algorithms based on Artificial Immune Systems.

It is important to note that none of the state-of-the-art AI approaches applied to the MP problem in the recent literature produces results that are remotely capable of approaching the quality of the algorithm (in its variations) presented here. The reasons for this have been discussed in [14] and are largely due to the usage of the offline error, which rewards early discovery of elevated locations. This cannot be achieved reliably with stochastic methods. All known contemporary approaches, however, rely heavily on stochastic elements.

III. ALGORITHM

The components of the algorithmic framework used for the comparison of the local search methods are explained at length in previous work [14]. As the global search is the same in both cases, its implementation is only of marginal interest. The complete algorithm is outlined below.

Algorithmic Framework

1. Set the initial base point \( x^0 \) to uniformly random coordinates. Search the space in even intervals in every dimension to create a group of candidates. Choose the best candidate \( x^{+1} \) as the basis of the local search.

2. Perform one of the local search options explained in sections A and B. If the locally optimised solution \( x^{+r} \) is not within a predefined distance of any other solution, save it.

3. Repeat steps 1 and 2 while the objective function remains unchanged.

4. When a change has occurred, reoptimize the saved solutions by applying the respective local search.

In this implementation, the fine-tuning phase, which was observed in [13] to contribute significantly to the quality of the result – largely due to the fact that it allowed the local search phase to stop short of using very small step lengths toward the end of the search, saving function evaluations – was omitted to establish the most favourable step sequence for each of the algorithms.

A. Hooke-Jeeves Pattern Search

This very simple local search was proposed by Hooke and Jeeves more than 40 years ago [10]. It is still among the first choices for researchers in need of a deterministic local search.

The most recent applications include Neri et al. [15] as well as Benasla, Abderrahim and Rahli [2]. The former used the Hooke-Jeeves Pattern Search (HJ) to optimise the design parameters of a controller for an electric engine, the latter to find the optimal power production for a given grid system. HJ has also been used to optimise the parameters for a controller of a heat exchanger [1].

In many cases, the HJ algorithm is used to supplant a linear program. When applied to a multi-modal landscape, it is prone to stagnation at the nearest local optimum. Therefore, HJ is often applied in combination with a global search strategy, such as Evolution Strategies [4]. In the current work, HJ has taken the place of the Dimensional Local Search (DLS) algorithm used in previous work with...
Exceptional success [14]. The HJ algorithm is listed below.

**Hooke-Jeeves Pattern Search Algorithm**

1. Obtain initial base point \( x^0 \). Determine set of step lengths.
2. Move the basepoint along every one of the \( d \) dimensional axes at a time and evaluate the result. Adopt each new point if it improves on the previous point. If any of the moves was successful, go to 3. If none was successful, goto 4.
3. Repeat the successful moves in a combined pattern move. If the new point has a better fitness, assume it as the new base point. Return to 2 whatever the outcome.
4. Adjust step length to next smaller step. If there is a smaller step, move, although successful, has “overshot the goal” and needs to retract. It is therefore useful to attempt a move in the opposite direction, but only when following the previous move’s direction has failed.

**IV. Step Lengths**

During the development of the MMEO algorithm, it became clear that the step lengths used for the local search are crucial for the performance of the local search, which is the largest contributor to the result quality. The sequence \( S = \{s_0, s_1, \ldots, s_m\} \) used for the MMEO algorithm was found to play a crucial part in its performance. It was first devised experimentally by choosing individual numbers for a sequence, the most successful of which ultimately described an exponential decline. The outcome coincides with pertinent approaches to local search methods in the literature, e.g. [16] which seem to agree on the superiority of an exponential function for step lengths in local search.

\[
s_j = s_{j-1} \times b^j \tag{2}
\]

The sequence obtained experimentally [14] can best be approximated by equation (2). The initial value \( s_0 \) is determined on the basis of knowledge about the search space, specifically the approximate distance a point has to move from a lowest point to the nearest local optimum, preferably erring on the shorter side, because the exponential decline dictates that many small moves have to make up for a large move which failed by a small margin. The sequence has reached its end \( s_m \) when the value drops below a predefined limit, which, again, is dependent on the problem space. Given the exponential pattern of the sequence, it is important to ensure that the last value \( s_m \) is indeed close to the desired smallest step length.

Consequently, the number of steps \( m \) in the sequence \( S \) will depend, besides on \( s_0 \), on \( b \), which is a value between 0.2 and 0.8. Different combinations of \( s_0 \) and \( b \) were explored. The initial assumption was that the local search would perform best if the step length sequence \( S \) started with a value slightly below the average length. This seems appropriate as the solution would have to travel from the slightly elevated position it was left at by the global search. Since the average width of the peaks in Scenario 2 is 6.5 and the average distance a peaks travels at each change is 5, values between 2 and 10 seemed appropriate for \( s_0 \).

**V. Experiments**

For the experiments, Scenario 2 [7] of the MP function was used with 10 peaks, the direction of the moves of the optima set to \( \lambda = 0.0 \) (random), the maximum distance of the moves to 10. Scenario 2 permits 5000 FE between the changes. Every experiment was run with 100 changes and repeated 50 times for a representative sample. These settings were used by most of the benchmark results available.

Scenario 2 is a problem instance with 5 dimensions. Given the quality of the results obtained from the MMEO
algorithm, the Scenario 2 settings listed above were also used for trials in 10, 15 and 100 dimensions to detect possible differences in the way the local search types advance.

Each experiment was run 50 times on the problem in the given dimensions. The experiment with 50 trials was repeated for different combinations of initial values $s_0$ and bases $b$ for the exponential function. For the base $b$, all values between 0.1 and 0.9 were explored in steps of 0.1. It became apparent that the most successful sequences are obtained with the base values 0.2, 0.3 and 0.5. Preliminary trials with initial step lengths $s_0$ suggested some benefit from extending the range to a maximum of 13. Having to apply such a wide range of step sequences is a consequence of randomising the width of the hills as prescribed by Scenario 2 [7]. Although a typical hill climbing problem well suited to benchmark the HJ algorithm, its drawback lies in the difficulty to find an optimal step length.

VI. RESULTS

The best results are clearly achieved using $b = \{0.2, 0.3, 0.5\}$. The results of both algorithms vary greatly depending on the initial value $s_0$ used. Moreover, the best results are obtained from different initial values over the trials in different dimensions. In Table I, the results have therefore been grouped according to the base value $b$, as there seems to be slightly more consistency over this parameter.

The results conclude in HJ’s favour in all experiments above 5 dimensions. The averaged standard deviations over the trials with 50 experiments each differ only in the second precision for all algorithms in 5, 10 and 15 dimensions. They are 0.1 for 5, 0.6 for 10 and 0.9 for 15 dimensions. In 100 dimensions, DLS has a slightly higher standard deviation of 7.7 compared to HJ’s 6.6.

Since the aim of this paper is to appraise the potential of HJ, the usage of its core feature, the pattern move (PM), is of interest. It appears that an average of 40% (varying between 37 and 43%) of all moves are PM, except in the case of 5 dimensions, where there are, on average, only 37%.

Table II lists the best results and the number of successful pattern moves used in the trials which provided the five best results. The highest and lowest numbers of successful PMs are given for the best and fifth best values achieved. Note that the 5 percentages of successful pattern moves are not in the same order as the 5 best offline errors.

The results average the number of successful pattern moves which are a percentage of the approx. 40% pattern moves made in a trial. The offline errors are given for the best and fifth best values achieved. Note that the 5 percentages of successful pattern moves are not in the same order as the 5 best offline errors.

VII. DISCUSSION

The results seem to indicate that there is little benefit from successful PMs. In lower dimensions, the numbers seem haphazard; in 100 dimensions, we seem to find that the

---

**TABLE I**

<table>
<thead>
<tr>
<th>Dimensions</th>
<th>Step length settings</th>
<th>HJ</th>
<th>DLS</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d$</td>
<td>$b$</td>
<td>$s_0$</td>
<td>min oe</td>
</tr>
<tr>
<td>5</td>
<td>0.2</td>
<td>0.35</td>
<td>0.68</td>
</tr>
<tr>
<td></td>
<td>0.3</td>
<td>0.26</td>
<td>0.62</td>
</tr>
<tr>
<td></td>
<td>0.5</td>
<td>0.29</td>
<td>0.71</td>
</tr>
<tr>
<td>10</td>
<td>0.2</td>
<td>1.60</td>
<td>1.91</td>
</tr>
<tr>
<td></td>
<td>0.3</td>
<td>1.39</td>
<td>2.25</td>
</tr>
<tr>
<td></td>
<td>0.5</td>
<td>1.16</td>
<td>1.93</td>
</tr>
<tr>
<td>15</td>
<td>0.2</td>
<td>2.31</td>
<td>2.97</td>
</tr>
<tr>
<td></td>
<td>0.3</td>
<td>2.23</td>
<td>2.79</td>
</tr>
<tr>
<td></td>
<td>0.5</td>
<td>2.09</td>
<td>3.07</td>
</tr>
<tr>
<td>100</td>
<td>0.2</td>
<td>14.70</td>
<td>18.92</td>
</tr>
<tr>
<td></td>
<td>0.3</td>
<td>17.60</td>
<td>21.53</td>
</tr>
<tr>
<td></td>
<td>0.5</td>
<td>17.79</td>
<td>22.63</td>
</tr>
</tbody>
</table>

**TABLE II**

<table>
<thead>
<tr>
<th>Dimensions</th>
<th>Number of Successful Pattern Moves Used in the Trials Which Provided the Five Best Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d$</td>
<td>Successful PM All Experiments</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>15</td>
<td>15</td>
</tr>
</tbody>
</table>

The results compare the Hooke-Jeeves and Dimensional Local Search algorithms. The three bases $b$ have been combined with every step length between 4 and 13, resulting in 9 different trials for every $b$. The results are very different over the 9 runs per base, the maximum and minimum offline errors (oe) are shown.
fewer successful PM, the better the results of the trials. The question then becomes, how can unsuccessful moves improve a local search. It may be useful to look at pattern moves in more detail.

A. Dissecting Hooke-Jeeves

Here we look at two step sequences in the algorithm which occur between two changes of $s_j$. To obtain a good illustration, it became necessary to deviate from standard flowchart notation.

In Fig. 1, the algorithm starts with $d=4$ moves, 2 of which fail. The failed dimension use 4 function evaluations (FE) to ensure that an advancement of length $s_j$ is not possible in either direction. The successful moves will, on average, take 3 – one of them will advance in the first direction tested, one in the second direction explored.

The dimensions that did not improve (2 and 4) are suspended until the next step length adjustment. The other two (dimensions 1 and 3) are advanced the same length in a single PM which uses a single FE. Typically for the step lengths chosen for our experiments, half of the moves combined in the PM will “fail”, i.e. they will not contribute to the increase in fitness.

The PM itself will fail if the fitness of the “successful” part-move has an increase which cannot compensate for the decrease in fitness caused by the coordinate in the “failing” dimension (in this case 3). Assuming this is the case, (as Fig. 1 does), the next EM will advance the dimension which contributed the fitness, and fail the one that did not. Consequently, the next PM will now only move the solution in dimension 1.

In Fig. 1, it is assumed that this PM fails. As the PM is not expected to provide any knowledge about the possibility of advance in a dimension – although, in this case, with a single variable, it could – the next EM is executed in both directions (using 2 FE) to establish the failure of the move in this dimension.

It is easy to see from this example that neither PM contributed to the result. However, their cost was only 2 FE, resulting in a total 14 FE for the sequence of moves.

Even if PM 2 had succeeded, there is no gain from using it, since the number of FE when changing a single dimension is always 1.

In Fig. 2, we have a similar situation to Fig. 1, except that in this case, PM 1 succeeds. As in the previous case, only one of the moves in the individual dimensions (dimension 1) contributes to an improvement. This time, however, the contribution is large enough to compensate for the fitness lost by the move in dimension 3. The variables in both dimensions are advanced. In the next EM, it is discovered that 1) no more moves can be made in dimension 1 and 2) that the move made in dimension 3 can be gainfully reversed. The reversal will always take 2 FE, as the algorithm always advances in the same direction first in order to save function evaluations.

Now that the reversal was successful, it leads to another PM, which is bound to fail, as it would actually take the variable in dimension 3 back to the state it was in before EM1. In this scenario, the pattern moves have led to an additional 3 FE in the development of variable 3.
The problem might be further exacerbated if the variable in dimension 1 could be improved for a longer period of time, leading to dimension 3 being "dragged along". More detailed studies of the optimising process reveal that if several “improvable” variables are involved in the PM, moving a variable in one dimension back and forth several times does indeed happen. The variable is moved the same step length and then reversed – and moved again in the reverse direction – until the other variables can no longer be improved.

If this happened in the scenario depicted in Fig. 2, PM 2 would change both dimension 1 and dimension 3. Dimension 3 would now be moving in the opposite direction. Since the negative effect of moving it forward has, in PM 1, been successfully covered by the improvement of dimension 1, it is now unlikely, though not impossible, that this PM would succeed. If the PM was unsuccessful, dimension 3 could now be suspended, while dimension 1 would be moved forward in the next EM. Since this would, at last, succeed, another PM would take place to move dimension 1 on its own.

To establish the extent of the problem, the number of reversed moves was recorded, using two experiments (10 and 100 dimensions) with 50 trials each. The trial in 10 dimensions yielded an average rate of 2 “reversed” variables per successful PM. In the same experiment, a successful PM moved the solution, on average, in 7 dimensions. In the experiment with 100 dimensions, a successful move changes, on average, 70 variables, 17 of which are subsequently reversed.

Both of the situations depicted in Fig. 1 and Fig. 2 would be handled the same way by DLS. Remember that the only difference between the two situations was that in the latter instance, PM 1 was successful because the fitness increase in dimension 1 compensated for the loss in fitness caused in dimension 3, whereas in the first example, it did not do this. The steps in Fig. 3 are simple enough to be self-explanatory.

These examples seem to prompt the assumption that the PM as an algorithmic feature must be more successful in higher dimensions, as the number of individual moves in DLS – or EM in HJ – increases. Quite surprisingly, the results in Table II indicate the opposite. To find out more,

The results of the test indicate that there is no significant difference between the sets in all three tests. However, there is a decline in the results produced by the same algorithm with different step length sequences $S$.

Interpreting these results, it may not be helpful to attribute too much weight to the outcomes in 5 dimensions, as the solvers may have reached a quality barrier that cannot be penetrated. In higher dimensions, omitting the pattern move seems to lead to some loss of quality. The difference between the best outcome of both algorithm versions, however, is far smaller than the variation in the results produced by the same algorithm with different step length sequences $S$.

To test the hypothesis of no difference between HJ using the pattern move and not using it, a paired t-test was used on all test runs in 10, 15 and 100 dimensions. The asymptotic significances returned by the Kolmogorov-Smirnov tests over all six datasets (offline error results in the given dimensions with and without pattern moves) lie between 0.2 and 0.9, exceeding the threshold of 0.05 by a large margin. Therefore, the six test series with 30 different combinations of base values (0.2, 0.3 and 0.5) initial values (4 through 13) are clearly normally distributed.

### Table III

<table>
<thead>
<tr>
<th>Dimensions</th>
<th>HJ</th>
<th>HJ, no PM</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d$</td>
<td>$\min \ o e \ b , s_0$</td>
<td>$\avg \ o e \ b , s_0$</td>
</tr>
<tr>
<td>5</td>
<td>0.26</td>
<td>0.3</td>
</tr>
<tr>
<td>10</td>
<td>1.16</td>
<td>0.5</td>
</tr>
<tr>
<td>15</td>
<td>2.09</td>
<td>0.5</td>
</tr>
<tr>
<td>100</td>
<td>14.70</td>
<td>0.2</td>
</tr>
</tbody>
</table>

### Table IV

<table>
<thead>
<tr>
<th>Dimensions</th>
<th>Difference between Pairs</th>
<th>95% Confidence Interval</th>
<th>Indicators</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d$</td>
<td>Mean</td>
<td>StdD</td>
<td>Lower</td>
</tr>
<tr>
<td>10</td>
<td>0.00</td>
<td>0.16</td>
<td>-0.7%</td>
</tr>
<tr>
<td>15</td>
<td>0.00</td>
<td>0.18</td>
<td>-0.8%</td>
</tr>
<tr>
<td>100</td>
<td>0.46</td>
<td>1.90</td>
<td>-1.2%</td>
</tr>
</tbody>
</table>

All paired tests were run with two sets of 30 values, one set with and one set without pattern move, hence there are 29 degrees of freedom (df). The results of the test indicate that there is no significant difference between the sets in all three tests. However, there is a decline in significance for the sets in 100 dimensions.

For the t-test, the trials with the same number of dimensions with and without pattern moves are paired using the base and initial values. The null hypothesis of no difference is supported by significance values that surpass the required level of 0.05 by several magnitudes. Given a correlation of 0.6 for each of the pairs in 10, 15 and 100 dimensions, we can safely conclude that there is no
B. Pattern Move and Step Lengths

The intuitive expectation is that HJ - with its pattern move - would perform better with a slightly shorter step length than DLS or HJ without a pattern move, simply because it reduces the average number of failed pattern moves. The only confirmation for this theory is the experiment with 100 dimensions.

For the pattern move to contribute significantly to the result outcome, the step lengths should, on average, be at least half the size of the distance which needs to be travelled. This is a conundrum, because if we knew exactly what distance the solution has to travel to reach the optimum, we would set the step length to exactly this measure, and not twice the measure for the sole purpose of accommodating an additional pattern move. The pattern move will, therefore, prove least helpful in areas where the step lengths – and hence the algorithm performance – are highly optimised, i.e. in the range where we obtain excellent results because the step length has been optimised to require, on average, a single step.

There is another consideration which pertains to the offline error as a performance measure. The offline error rewards early submission of good results, therefore the initial step $s_0$ - and to some extent the second largest step $s_1$ - are of crucial importance. $s_0$ can still be set relatively precisely, as the search space usually yields some indication as to how far the solution might have to travel. The subsequent steps are, however, dependent on the first ones, and harder to determine. Fortunately, because of the small advancement they entail, they bear relatively little responsibility for the quality of the offline error. It might therefore be interesting to test the HJ pattern move in an environment where the performance is measured differently.

VIII. CONCLUSION

It has been observed that the pattern move of the HJ algorithm has little effect on the result quality when solving the Moving Peaks benchmark. Insights into the pattern move’s effects in concrete situations suggest that this phenomenon might generalise to a larger number of problems. The effectiveness of the pattern move is especially questionable in areas of superb result quality, i.e. in areas where the local search step lengths have been optimised and where the result quality relates to the number of function evaluations used.

The investigations reported here suggest that even if the pattern move could be used gainfully, it would be advisable to “switch it off” when only one dimension is left to be optimised and all others have been suspended until the next adjustment of step length.