Randomising Block Sizes for BlockCopy-based Wind Farm Layout Optimisation

Michael Mayo, Maisa Daoud and Chen Zheng

Department of Computer Science, University of Waikato
Hamilton, New Zealand
mmayo@waikato.ac.nz, {mttd1,cz90}@students.waikato.ac.nz

Abstract. The BlockCopy stochastic local search algorithm is a state-of-the-art optimiser for the Wind Farm Layout Optimisation problem. Unlike many other metaheuristics-based optimisers, BlockCopy requires the specification of only one key parameter, namely a block size. In this paper, we investigate the effect on different block sizes on the optimisation results. Using standard benchmarks for the Wind Farm Layout Optimisation problem, we show that smaller fixed block sizes (relative to overall layout size) produce better optimised layouts than larger fixed block sizes. More interestingly, we also show that randomising the block size parameter results in optimisation performance at the same or a better level than that produced by the best algorithm with a fixed block size. Effectively, this means that the user can ignore the need to tune the block size parameter and simply randomise it instead. Such a strategy results in what is effectively a parameterless, but none-the-less effective, optimisation algorithm for the Wind Farm Layout Optimisation problem.

Keywords: wind farm layout optimisation, blockcopy, local search, parameter tuning

1 Introduction

The Wind Farm Layout Optimisation problem concerns finding the optimal positions for wind turbines (termed "micro-siting" in the literature) in a planned wind farm [7–9, 5]. In a wider context, the process of micro-siting occurs after (i) the site and boundaries for the wind farm has been chosen, (ii) the characteristics (i.e. typical distributions across wind speeds and directions) for the site have been measured, and (iii) the make, model and other details of the particular turbines (which are manufacturer-dependent) have been chosen.

Micro-siting is critical in wind farm design for several reasons. The primary reason, which happens to be the one focussed on in this work, is the phenomenon of wake interference: if two or more adjacent turbines are incorrectly sited, then the wake (i.e. the reduced velocity downstream wind) of one turbine can unduly negatively influence the other turbines. This negative influence manifests as reduced downstream power generation or, alternatively, as an increased probability of damage to downstream turbines due to increased turbulence.

The Wind Farm Layout Optimisation problem therefore addresses the problem of turbine micro-siting in order to mitigate these negative effects of wake interference.

In terms of practical relevance, solving this problem is of immense importance to the global wind industry – and therefore the world economy in general. Wind farms worldwide are typically becoming larger and larger (for example, the London Array [1] generates 630MW of power that could power 490,000 households) and therefore small efficiency gains in design may lead to power for significantly more households.

Conversely, the problem is also of considerable interest to researchers in metaheuristics and related fields. This is for two reasons. Firstly, the solutions to the problem are fundamentally multi-dimensional rather than one dimensional as is typically the case for many blackbox test functions. In the simplest case, a wind farm can be represented as a two-dimensional (flat) layout with turbine positions specified using two coordinates x and y. In more complex cases, additional attributes such as turbine height further increase the dimensionality of the problem. Since wake effects are non-linear and concern interactions between neighbouring turbines, then correspondingly the optimisation problem is non-linear with considerable epistasis between the variables being optimised.

The second reason why the Wind Farm Layout Optimisation is of interest to researchers is because it represents a problem with a significantly complex evaluation function. Since wake effects between every pair of turbines must be computed individually, the time complexity of the simplest wake models is at least $\mathcal{O}(n^2)$ where n is the number of turbines. The makes the evaluation of a significant quantity of potential layouts infeasible, and therefore many methods fail to achieve satisfactory results.

In this paper, we continue our exploration of a recently proposed stochastic local search method for optimising wind farm layouts. The method, known as BlockCopy [6], divides a wind farm layout into square regions and performs stochastic local search by copying blocks of turbines from one region to another.

Previously, we were able to show that BlockCopy significantly outperforms another state-of-the-art algorithm called the Turbine Displacement Algorithm [10].

In this paper, we address one significant issue that arises immediately when the BlockCopy algorithm is used: how large should the blocks be? Our findings indicate that block size has a significant impact on the optimisation performance of the algorithm when the block size is fixed throughout a single run of the algorithm. However, if the algorithm is modified so that instead of a fixed block size, the search operator instead picks a random block size every time it is applied, then the approach performs as well as (or in some cases, better than) the best approach with a fixed block size. The net effect of the finding reported here is that the user is obviated from the need to select a block size, and the BlockCopy local search algorithm effectively becomes parameterless.

2 Background

2.1 Wind Farm Layout Optimisation Problem

In its simplest form, the Wind Farm Layout Optimisation belongs to the family of two-dimensional circle-packing problems: i.e. the problem is to find optimal positions of a set of points (x, y) – each point representing a turbine in our case – such that no two points lie too closely together and all points lie inside a pre-specified 2D shape.

Unlike typical circle packing problems, however, the objective in the wind farm case is not to maximise the number of packed circles but instead to maximise either the total energy efficiency of the farm or to minimise the cost of energy produced by the farm. The minimum distance constraint between the points represents the minimum inter-turbine distance: if turbines are placed to closely together, then wake effect models generally become inaccurate [9] and layouts cannot therefore be evaluated properly. Following Samorani [8] we set the minimum turbine distance to 120m, or three times the rotor diameter for turbines with a 40m diameter rotor.

In this work, we use the Jensen far wake model [2, 4] to evaluate potential layouts. The Jensen wake model is an analytic approach for assessing wake interferences for two-dimensional layouts where the distribution of potential wind speeds and directions at the farm site is assumed a constant.

Although originally proposed in the mid-1980s , the Jensen is still used widely in the community. To illustrate, Samorani [8] describes it precisely in a recent 2013 introductory survey on the Wind Farm Layout Optimisation problem, and Shakoor et al. [9] performed an extensive comparison of several different wake effect models, concluding that "... Jenson's far wake model is a good choice to solve the wind farm layout optimisation problem due to its simplicity and relatively high degree of accuracy."

A graphical illustration of the wake effect according to the Jensen model is given in Figure 1. To explain briefly, incoming wind travelling at speed u_0 m/s reaches a turbine with rotor radius r_r. As the wind passes through the turbine's blades, its velocity is reduced. At a distance of x meters downstream, the wind speed is u_j m/s, which must be less than u_0. Moreover, the wake radius increases linearly in size with distance. At x meters downstream, its radius becomes $r_1 > r_r$.

The exact mathematical details of the Jenson wake model, including how the reduced wind speeds and the spreading wake radius should be calculated, are adequately explained by Samorani [8]. We follow the exact same approach in our implementation, and therefore the interested reader is referred to Samorani [8] for the specific mathematics behind the model (which we do not present here for reasons of space).

It suffices to say, however, that one key characteristic of the Jensen model is that the wind velocity deficit caused by a turbine is inversely proportion to the distance x squared from the turbine that caused it; therefore, eventually, most wakes become negligible for large enough x. However, if a turbine lies

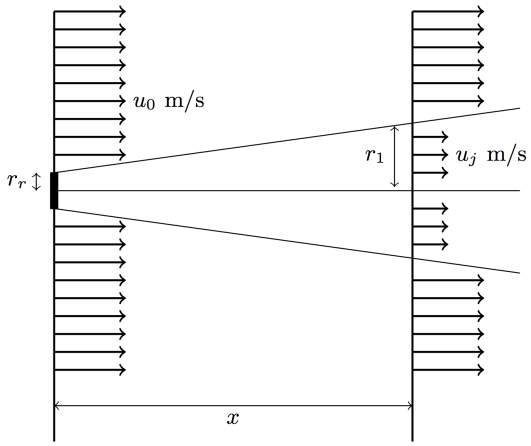


Fig. 1. Depiction of the wake effect (reproduced from [8]).

inside multiple wakes (which is highly likely for small, dense layouts) then the wakes aggregate and so even small wake effects must be calculated. Again, the interested reader should refer to Samorani [8] for the method of aggregating wakes.

One important aspect of the Jensen model is how a turbine's power output is computed. Generally speaking, such a calculation depends on the type of wind turbines being installed and is therefore manufacturer-specific. Therefore, in this paper, we used an idealised turbine model described originally in Mosetti et al. [7] and later described again by Samorani [8]. In this model, the power output of a single farm is proportional to the cube of the incoming wind speed between two boundary wind speeds. The boundary wind speeds are known as the cut-in speed and the nominal speed. If the wind speed increases beyond the nominal wind speed, then power generation becomes a constant until a cut-out speed is reached. For turbines with a cut-in speed of 2 m/s, a nominal speed of 12.8 m/s and a cut out of 18 m/s, then the power curve used to model individual turbines is:

$$power(u) = \begin{cases} 0\text{kw} & \text{where } u < 2\text{m/s} \\ 0.3u^3\text{kw} & \text{where } 2\text{m/s} \leq u < 12.8\text{m/s} \\ 629.1\text{kw} & \text{where } 12.8\text{m/s} \leq u < 18\text{m/s} \\ 0\text{kw} & \text{where } u > 18\text{m/s} \end{cases} \quad (1)$$

This is precisely the wind turbine model that we use in this research.

Finally, we briefly turn to the objective function for the Wind Farm Layout Optimisation problem. We have already alluded to the fact that it is focussed on minimising the wake effect. More precisesly, if $\mathbf{l}$ is a layout, then the objective function can be written down as:

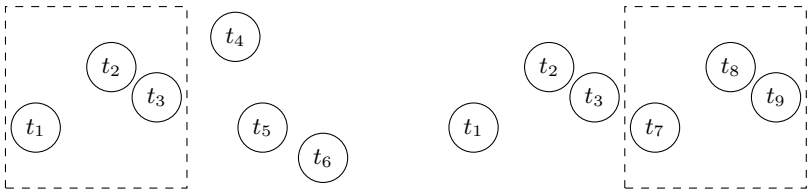
$$F(\mathbf{l}) = \sum_{s \in S} r_s \frac{\sum_{j \in \mathbf{l}} power\Big(u_s(1 - v^s_{def}(j))\Big)}{\sum_{j \in \mathbf{l}} power(u_s)} \tag{2}$$

where j is a turbine's position in the layout, and $v^s_{def}(j)$ is the total velocity deficit (i.e. total loss in power as a proportion between 0 and 1 due to wake interference) at j. Th total velocity deficit is a quantity aggregated across all other turbines in the layout that may be affecting the turbine at position j. S is a set of wind scenarios, in which each element of S is a potential wind speed/direction pair. This must be measured at the wind farm site and is considered to be a constant across the entire layout. The variable r_s represents the probability of scenario $s \in S$ and u_s is the specific wind speed under scenario s. It should be evident that $\sum_{s \in S} r_s = 1.0$ in order to compute proper expected power values.

The objective value of a layout, therefore, is defined as the total power output of the farm *with* wakes divided by the total, potential, power output of the farm *without* wakes. Clearly, this is a ratio that is maximised at 1.0, and therefore our aim is to find layouts with an objective value as close to 1.0 as is possible.

2.2 BlockCopy-based Stochastic Local Search

The BlockCopy stochastic local search algorithm, first described by Mayo and Zhen [6], optimises wind farm layouts by defining a search operator that copies entire groups of turbines at a time from one place on the layout to another. It iteratively applies this operator, starting with a randomly-generated layout called the "current" layout, to generate a progression of new layouts. The current layout always represents the fittest layout found so far. If a new layout has improved fitness compared to the current layout (as a result of one block copy operation), then the new layout becomes the new "current" layout of the search. Otherwise, if the new layout has a lesser fitness, it is discarded.



(a) Initial layout with selected source block.

(b) Final layout showing new target block.

Fig. 2. Illustration of the BlockCopy operator. In this example, the left block of a small layout is duplicated to the right hand side of the layout.

The basic search operator is illustrated in Figure 2. In our current formulation of the algorithm, the regions that are copied are square "blocks" of a fixed size

such that the layout can be divided into blocks that do not overlap and that exhaustively cover the entire layout.

One application of the BlockCopy operator proceeds by randomly selecting a source and destination block. The turbines in the destination block are deleted, and then turbines from the source block are copied to the destination block one at a time. Such an approach basically preserves the local configuration of turbines in the source block.

After the copy operation is complete, if the total number of turbines in the layout has either increased or decreased (because of differences in the number of turbines present at the source and destination block), then turbines are either randomly added or randomly purged from the layout globally in order to keep the total number of turbines in the layout at a constant value.

All constraints must also be adhered to by the operator so that new layouts are always legal: if the copying of any turbine would result in a constraint violation, then it is simply not copied.

The reasoning behind this approach is that by maintaining the relative configuration of turbines whenever a block is copied, then if a particularly good local configuration of turbines is present in the layout, then this configuration will quickly replicate itself across the layout via successive BlockCopy operations.

In the initial recently published evaluation of the algorithm [6], it was shown that BlockCopy outperforms one state-of-the-art approach called the Turbine Displacement Algorithm [10] on a set of benchmark problems using a cost-based objective function and a different far wake model than Jensen.

Moreover, it was also shown in the previous paper that the local search variant of BlockCopy (with only a single current layout held in memory) outperformed a population-based evolutionary variant also using the BlockCopy operator for mutation and/or crossover. We believe that the primary reason for this is the limited number of evaluations that the problem affords: local search effectively performs significantly more exploitation of the search space than population-based approaches do. Population-based strategies, in contrast, are better balancers of exploitation and exploration. The cost of this, however, is that more evaluations are required to reach the same level of fitness. We believe this makes local search a better option given the time complexity of the evaluation function.

A significant disadvantage of the BlockCopy approach, however, is that it is not clear what the best default size for the blocks should be. We therefore conducted the set of experiments described in the next section to assess the impact about different block size decisions on the optimisation results.

3 Experimental Setup

In order to perform experiments related to Wind Farm Layout Optimisation, one must firstly define and/or obtain some benchmark problems so that any results can, in the future, be compared with other results from the literature.

In this paper, therefore, we assume that the layout is comprised of a fixed quantity of 64 turbines, and that this number can neither increase nor decrease.

Furthermore, we also assume that the layout is a square of size 1.5km × 1.5km. Turbines cannot be placed outside of the layout boundaries. Fixing the layout to such a small size effectively causes the wake effects to become a non-trivial negative influence on the wind farm's efficiency, which makes the problem interesting.

The optimisation algorithms are all initialised with different random starting layouts. The algorithm for constructing a random starting layout is straightforward: turbines are iteratively added at random locations on the layout, as long as they do not cause any constraint violations (i.e. the new turbine must not be placed too closely to an already-placed turbine). If a constraint would be violated by a placement, then a new random position is chosen for the current turbine. This is repeated until all 64 turbines have been initially placed.

Next, we utilise Samorani's [8] three different problems for benchmarking wind farm layout optimisation algorithms. The benchmark problems are defined by Table 1.

Problem	Direction(s)	Expected Speed(s)	#Wind Scenarios
A	$\{0°\}$	{12m/s}	1
B	$\{0°, 10°, \ldots, 350°\}$	{12m/s}	36
C	$\{0°, 10°, \ldots, 350°\}$	{8m/s,12m/s,17m/s}	108

Table 1. Problems from Samorani [8].

Problem A is the simplest benchmark, and consists of only a single wind scenario in which wind blows with uniform expected speed and in a single direction. The set of scenarios S for the objective function therefore consists of only a single element.

Problem B, alternatively, consists of 36 different wind scenarios. Each scenario differs only in the wind direction. while the expected wind speed is a constant. Unlike Problem A, therefore, this benchmark has no single dominant direction.

Problem C is the the most interesting and challenging of the three benchmarks. As is the case with Problem B, there are 36 possible wind directions. In Problem C's case, however, for each different wind direction, there are also three different expected wind speeds. Furthermore, there is a clear dominant wind direction: 310° is the direction with the the highest probability of the greatest wind speed, and therefore it is also the direction of the greatest power production. In total, Problem C consists of 108 different wind scenarios.

Samorani [8] describes Problem C graphically by means of a histogram of wind speeds vs. directions. In order to implement this benchmark, we therefore reverse-engineered the probabilities from his publication. The probabilities we used for Problem C are given in Table 2.

Now that the three benchmark problems have been described, we next describe the variants of the BlockCopy local search algorithm that we tested.

Direction	$u_s = 8$m/s	$u_s = 12$m/s	$u_s = 17$m/s
0°-260°	0.00404	0.00865	0.0115
270°	0.00404	0.0107	0.0127
280°	0.00404	0.0121	0.0156
290°	0.00404	0.0141	0.0185
300°	0.00404	0.0138	0.0300
310°	0.00404	0.0190	0.0352
320°	0.00404	0.0138	0.0300
330°	0.00404	0.0141	0.0185
340°	0.00404	0.0121	0.0156
350°	0.00404	0.0107	0.0127

Table 2. Probabilities used for the 108 wind scenarios under Problem C (rounded to three significant figures) derived from a chart in [8].

The basic difference between the variants is how the size of the block is chosen. We selected four sensible fixed sizes for the blocks that were chosen because they divide the 1.5km × 1.5km layout evenly. The fixed sizes were: 125m (which divides the layout into 12×12 blocks), 250m (dividing the layout into 6×6 blocks), 500m (making 3×3 blocks) and 750m (which is 2×2 blocks). Each fixed size corresponds to one algorithm variant. We also tested a fifth algorithm which selects a block size from the above set of four sizes at random each time it performs a BlockCopy operation. We call this algorithm simply "random".

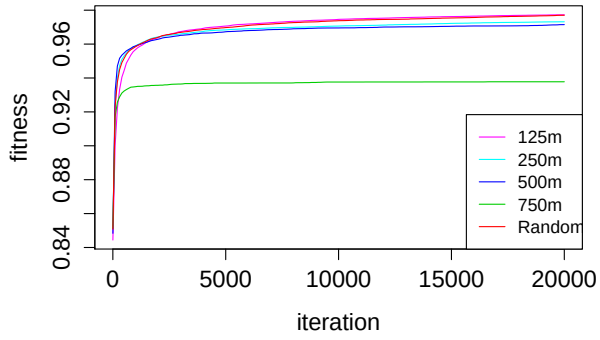
In order to obtain statistical results, we ran each algorithm 30 times on each benchmark. Each algorithm was run for 20,000 iterations before terminating. Therefore the total number of experimental runs performed was 3 benchmarks × 5 algorithms × 30 repeats, or 450 runs in total.

4 Results

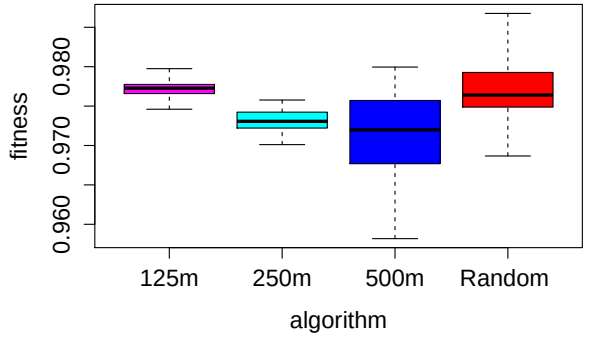
The results of our experiments are given in Figures 3, 4 and 5 for each of the problems A, B and C, respectively. Each figure has two parts: a subfigure (a) showing the convergence curves for each of the five algorithms (averaged across 30 runs), and a box-and-whiskers plot (b) showing the distribution of final results for each algorithm.

Examining the convergence curves firstly, we can see that the choice of block size has a significant impact on convergence performance. In particular, the very large block sizes (750m for Problem A and 500m+ for Problems B and C) result in significantly worse convergence curves. This is to be expected since larger block sizes correspond to larger steps in the search space – and such large steps may miss nearby local optima. Conversely, for the small block sizes (125m and 250m) the convergence curves are all fairly similar.

Figures 3(b), 4(b) and 5(b) depict the final distribution of fitness values after each of the 30 runs. Generally speaking, larger fixed block sizes clearly shift the median fitness down compared to smaller fixed block sizes. This negative effect

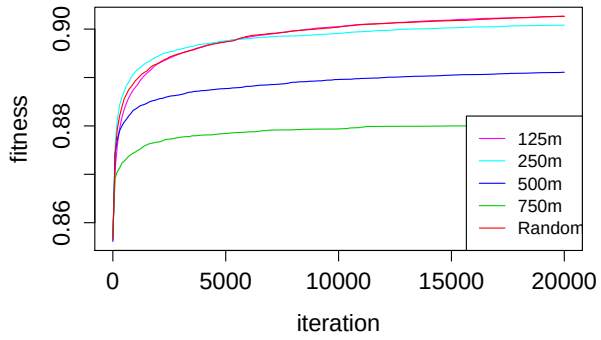


(a) Averaged convergence curves.

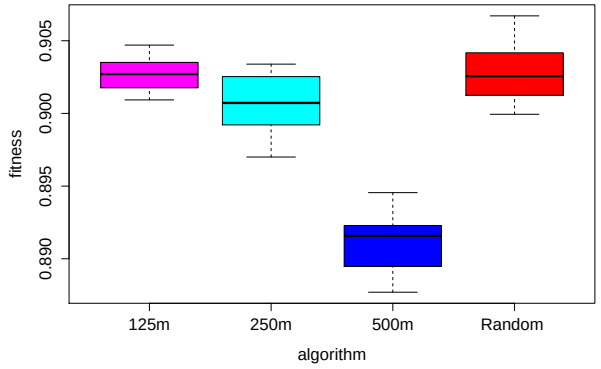


(b) Final fitness distributions.

Fig. 3. Results for Problem A (see online colour version for best viewing).

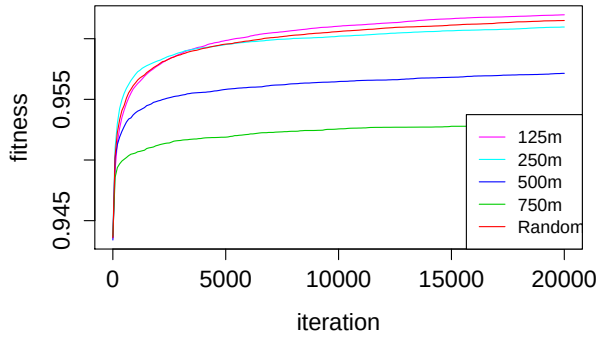


(a) Averaged convergence curves.

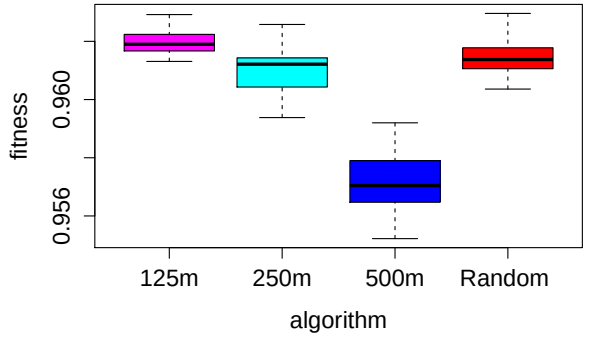


(b) Final fitness distributions.

Fig. 4. Results for Problem B (see online colour version for best viewing).



(a) Averaged convergence curves.



(b) Final fitness distributions.

Fig. 5. Results for Problem C (see online colour version for best viewing).

is most dramatic for the algorithm with a fixed block size of 750m, which also consistently has the worst convergence curve. This algorithm's fitness distribution is so low that that its particular set of results have been excluded from the plots so that the rest of the results can be read clearly. The worsening effect of larger block sizes is clearly apparent for the 500m block size when inspecting the box plots.

In terms of answering the question of which is the best overall fixed block size, the answer to that question appears to be 125m – which is also the smallest fixed block size considered. In all cases, the median and maximum fitnesses achieved by the 125m variant exceed those of all the other fixed block size algorithms.

The exception to the trend, however, is the random block size algorithm. Interestingly, this algorithm has quite a different behaviour to the fixed-size algorithms.

Firstly, it is always competitive with the other algorithms. In terms of median overall performance, it always performs slightly below the 125m fixed algorithm's median, but this performance is obviously not statistically significant due to the close overlap of the distributions.

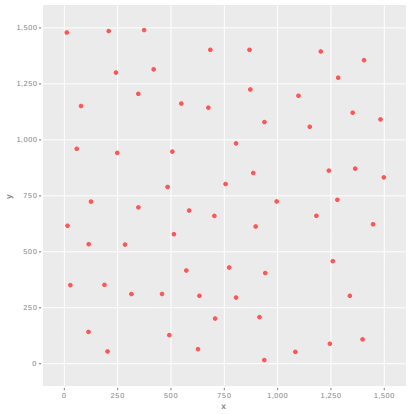
Most interestingly, however, the random algorithm's maximum (as opposed to median) achieved fitness over all runs turns out to be the greatest for each of the benchmark problems. The explanation for this appears to be largely due the random algorithm having a greater variance in final fitness results. This is increased variance is especially evident in the box-and-whisker plots for Problems A and B. It is also somewhat evident visually in the Problem C plots. As a consequence of this higher variance, over thirty runs, the random algorithm always finds the best single layout compared to the 125m algorithm. This difference is quite small for Problem C, but an inspection of the numeric results used to generate the plot indicate that the random algorithm does indeed produce the overall best layout.

To conclude this section, some of the best optimised layouts found by the random algorithm are presented in Figure 6. By way of contrast, the layouts are shown alongside one example of a random unoptimised layout.

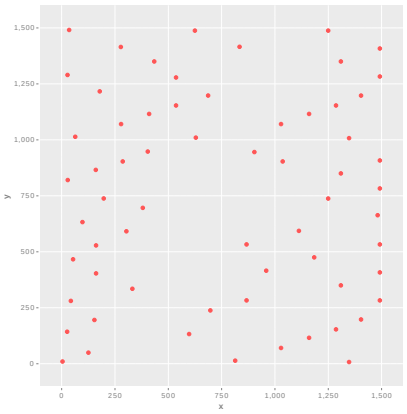
5 Conclusion

Overall, the results show that the randomised block size algorithm is a better choice than the algorithms that have a fixed block size, as long as the goal is to find the single best layout over multiple runs of the BlockCopy local search algorithm. Such a multiple-restart approach is in fact a sensible approach to take for most difficult optimisation problems.

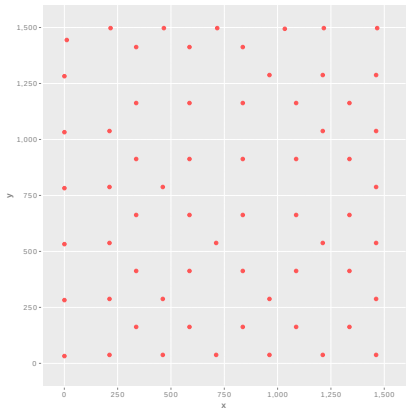
Moreover, the fact that the random algorithm is competitive with the best algorithm using a fixed block size means effectively that the user can bypass the problem of parameter tuning. Instead, the algorithm needs simply to create a set of appropriate block sizes for random block size selection. The tuning and selection of parameters is known to have a significant impact on the performance of metaheuristic optimisation algorithms [3]. By-passing this problem by simply



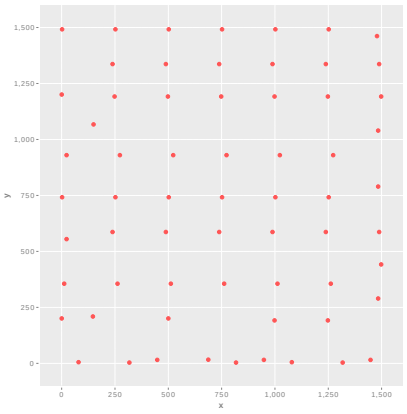
(a) Randomly generated layout.



(b) Best optimised layout found for Problem A.



(c) Best optimised layout found for Problem B.



(d) Best optimised layout found for Problem C.

Fig. 6. Example unoptimised (a) and optimised (b)-(d) layouts

randomising the main key parameter of the algorithm has been shown to be effective.

To conclude, future work will continue address the problem of improving the behaviour of the BlockCopy local search algorithm for the Wind Farm Layout Optimisation problem.

References

1. London Array brochure. Online PDF brochure, retrieved 9 Nov 2015, `http://www.londonarray.com/wp-content/uploads/London-Array-Brochure.pdf`
2. Jensen, N.: A note on wind generator interaction. Tech. rep., Risø DTU National Laboratory for Sustainable Energy (1983)
3. Karafotias, G., Hoogendoorn, M., Eiben, A.E.: Parameter control in evolutionary algorithms: Trends and challenges. IEEE Transactions on Evolutionary Computation 19(2), 167–187 (2015)
4. Katic, I., Høstrup, J., Jensen, N.: A simple model for cluster efficiency. In: Proc. Europe and Wind Energy Association Conference and Exhibition (1986)
5. Mayo, M., Daoud, M.: Informed mutation of wind farm layouts to maximise energy harvest. Renewable Energy 89, 437–448 (2016)
6. Mayo, M., Zhen, C.: Blockcopy-based operators for evolving efficient wind farm layouts. In: Proc. IEEE World Conference on Computational Intelligence, WCCI. p. to appear (2016)
7. Mosetti, G., Poloni, C., Diviacco, B.: Optimization of wind turbine positioning in large wind farms by means of a genetic algorithm. Journal of Wind Engineering and Industrial Aerodynamics 51(1), 105–116 (1994)
8. Samorani, M.: The wind farm layout optimization problem. In: Pardolas, P. (ed.) Handbook of Wind Power Systems, pp. 21–38. Springer-Verlag (2013)
9. Shakoor, R., Hassan, M., Raheem, A., Wu, Y.: Wake effect modelling: A review of wind farm layout optimization using Jensen's model. Renewable and Sustainable Energy Reviews 58, 1048–1059 (2016)
10. Wagner, M., Day, J., Neumann, F.: A fast and effective local search algorithm for optimizing the placement of wind turbines. Renewable Energy 51, 64–70 (2013)