Optimisation and Generalisation: Footprints in Instance Space

David Corne and Alan Reynolds

School of MACS, Heriot-Watt University, Edinburgh, UK
d.w.corne@hw.ac.uk, a.reynolds@hw.ac.uk

Abstract. The chief purpose of research in optimisation is to understand how to design (or choose) the most suitable algorithm for a given distribution of problem instances. Ideally, when an algorithm is developed for specific problems, the boundaries of its performance should be clear, and we expect estimates of reasonably good performance within and (at least modestly) outside its 'seen' instance distribution. However, we show that these ideals are highly over-optimistic, and suggest that standard algorithm-choice scenarios will rarely lead to the best algorithm for individual instances in the space of interest. We do this by examining algorithm 'footprints', indicating how performance generalises in instance space. We find much evidence that typical ways of choosing the 'best' algorithm, via tests over a distribution of instances, are seriously flawed. Also, understanding how footprints in instance spaces vary between algorithms and across instance space dimensions, may lead to a future platform for wiser algorithm-choice decisions.

1 Introduction

The chief purpose of research in optimisation is to understand how to find the most suitable algorithm for a given space of problem instances. When researchers concentrate on algorithm design without reference to specific problems, this view implies that the outcomes of such research should provide salient results on the applicability of the designed algorithms in terms of distributions of instances in one or more problem domains. Similarly, when research is clearly tied to a problem domain, the ideal outcomes would include a characterisation of the space of instances in that domain for which the studied algorithms are favoured.

We note that all of these desired outcomes are rare. However, such outcomes are extraordinarily important for two reasons: (a) it is easy to imagine practitioners choosing an algorithm for their problem on the basis of a research paper, and then applying it to real problems from a quite different instance distribution to that examined in the paper (indeed, often the paper's distribution is a small set of specific instances); (b) despite c. 50 years of research in optimisation, there is little re-usable knowledge that aids *a priori* choice/configuration of algorithms based on knowledge of the target distribution of instances.

Here we contribute to and explore the issues in (a) and (b) by examining algorithm 'footprints'. Such a 'footprint' indicates how an algorithm's comparative performance generalises in instance space (with reference to a collection of

algorithms under consideration). In particular, we find much evidence to support the claim that the typical ways of choosing the 'best' algorithm, via tests over a distribution of instances, are seriously flawed: this may rarely yield an algorithm that is best on many well-defined subsets within that instance space, and similarly outside that instance space. Also, algorithms have footprints in instance space that define the regions of prowess of certain configurations. Understanding these footprints, how they vary between algorithms and across instance space dimensions, may lead to a future platform for wiser algorithm-choice decisions.

The remainder is set out as follows. After discussing related work and associated issues in Section 2, Section 3 simulates the task of choosing an algorithm for a given space of instances, for each of two problem domains. After establishing the 'best' algorithm, by sampling from an instance space, we then partition instance space into subspaces and do further experiments on samples from these subspaces, in turn summarising the results by showing that individual algorithms have performance 'footprints' over the instance subspaces. This reveals, for example, the extent to which the presumed 'best' algorithm is not best for most of the instance subspaces. We discuss the implications of this in discussions within section 3, and conclude in section 4.

2 Related Work and Associated Issues

There is a rich academic literature concerned with understanding how to discover the ideal algorithm for a given problem. This broad topic encompasses many inter-related questions. High-level such questions include how we are to define both 'best algorithm' and 'given problem'. In different circumstances, the best algorithm may be defined in terms of mean quality of solution, speed of convergence, or some function of these and/or other factors. The given problem may be a suite of test problems from one or more domains, or a stochastically defined space of instances.

Given some appropriate context that disambiguates these high level questions, most research in algorithm design and configuration focuses on algorithms for configuration search. In such work, it is common to use a 'training set' of instances, and treat algorithm configuration as an optimisation problem in which we seek to optimise a measure of performance over this set. In some cases, the optimally configured algorithm's performance is validated on a test set. However the latter is quite rare in many factions of the literature on optimisation algorithms, in which the typical approach is to test algorithms on a suite of instances commonly used in the literature, with little or no consideration given to estimating performance on different instances.

Recently prominent in algorithm configuration research are Hoos and coworkers [1,2,3], who have designed stochastic search algorithms such as paramILS [4], which is very successful at finding ideal configurations for a given algorithm and set of problem instances. ParamILS is one of several approaches that have been independently investigated for this task (e.g. [5,6,7,8]), and is distinguished from others by its attention to the problems of 'over-confidence' (performance

on the training instances becoming an over-optimistic estimate of performance on unseen instances) and 'over-tuning' (the familiar machine learning scenario, in which prolonged training will lead to worse performance on the test set).

Essentially, such work (and similar, e.g. [9,10,11]) finds an algorithm that works best (given some definition) on a given instance distribution. Often this is a point distribution - a specific set of instances that may or may not be divided into training and test sets. It is worth noting that, though [4] deals with issues of over-confidence and over-tuning, such work rarely considers generalisation outside or within the instances considered. That is, the optimised configuration is delivered with (in [4], for example) some justified confidence that it is close to ideal for the training instances. However, it remains open how we might estimate its performance on alternative instances, or indeed on strict subsets of the training instance space.

To gain intuition for the significance of strict subsets of the instance space, consider configuring an algorithm for a job shop based on a given distribution of instances. By either using these instances directly, or constructing a generator based on them (e.g. with a distribution of processing times inferred from the instances), tuning will typically be biased towards some specific algorithm B, with an ideal mean performance over this instance space. However, there is an arbitrarily large number of coherent subsets of this space (defined by, for example, a given mean task length, or a given value for the range of processing times, and so forth) on which B's performance may well by trumped by an alternative configuration. The extent to which B may under-perform on new instances could be dramatic. When generalising outside the instance distribution, standard lessons from machine learning lead us to expect that, the better the performance of B on the test set, the worse its performance may be on unseen out-of-distribution instances. Although Hutter et al [4] avoid over-fitting in appropriate versions of ParamILS, this applies to the given instance distribution, and provides no certificates about performance outside this distribution.

Another area of related work is 'per-instance tuning', in which models are developed which enable an algorithm to be tuned based on features of the given instance [12–16]. In such work, typically preliminary experiments are done to capture the performance of several configurations of an algorithm on several instances. A model (e.g. perhaps a neural network) is then inferred, which predicts performance characteristics from combined instance features and algorithm parameters. Given a new instance, this model can then be used to predict the performance of any given configuration, and indeed a simple search can be wrapped around this, yielding predictions of ideal parameters for that instance. Smith-Miles et al [17] explore a variant of such ideas, concerned with algorithm choice based on instance features. In [17], 75,000 instances of a single-machine earliness/tardiness problem are generated, and on each instance they compare the earliest-due-date heuristic (EDD) and the shortest-processing time heuristic (SPT). A selection of learning methods are then applied, to infer models that relate instance features (e.g. mean due date, range of processing times, etc.) to the choice of either EDD or SPT. Results were very promising; some rules in-

ferred from a training set of instances could predict with c. 97% accuracy the best choice between these two heuristics on a test set. In turn, this gives promises for better-informed choice of algorithm than may be obtained, say, by resorting to the single algorithm that was best over a wide distribution of instances that included the instance in question.

Our arguments and contributions are complementary to such findings from the work so far on the per-instance tuning approach. The latter studies typically reveal that small differences between instances correspond to differences in the ideal algorithm for those instances. By arguing (and confirming in test domains) that an algorithm tuned over a space of instances may rarely be the ideal algorithm for subsets of that space (let alone outside that space), we deliver similar findings from an alternative perspective. However our emphasis is different, focusing on the particular subset of instances that is best addressed by a given algorithm. We argue that the nature of this 'footprint' is particularly interesting to study, and understanding how the footprints of individual algorithms vary (e.g. simulated annealing versus population-based search) may lead to new ways to choose the right algorithm for a given problem domain, especially where the model development costs of accurate per-instance tuning (for example) are infeasible due to problem size and associated concerns regarding computational expense.

3 Empirical Examples

3.1 Overview of Experiments

In the sequel, empirical footprints in instance spaces are produced for a selection of algorithms and two domains. The domains are single-machine job shop (SMT; optimising tardiness) and vehicle-routing (VRP; optimising a weighted combination of distance, tardiness and overtime). These were chosen partly for convenience (the SMT is simply implemented and evaluation of solutions is computationally light) and partly for relevance: both are very common in industry, and are such that individual sources of instances (e.g. factories, warehouses, distributors) can be expected to draw their real-world instances from individually characteristic instance spaces. Further domains (e.g. the quadratic assignement problem) are under study, but space restraints here limit us to two.

Our experiments had the following overall shape. A set A of algorithm variants is defined, and a distribution $p(I)$ of instances is defined over a space of instances I. Simulating a simple algorithm-choice scenario, each algorithm in A is applied (with n independent trials) to a sample from $p(I)$, and we record the algorithm A_{best} that achieves the best overall performance. To measure performance we consider both the fitness values reached after the maximum allowed time (a given number of fitness evaluation), and the time (evaluation number) in each trial when the best fitness of that trial was found. The details of defining 'best' are as follows: given the $|A| \times n$ results for each sample instance, we consider all pairs of results from distinct algorithms in distinct trials on the same instance; in each such pair, the algorithm with the better fitness gains a point,

with ties broken by speed of finding the solution. These points are accumulated over several instances. The algorithm with most points over a set of instances is considered best for that set, and the significance of its peak position (assessed by randomisation testing [18]) is considered in context. We found this more reliably inforative than simply resorting to mean fitness values per algorithm (which were not sfficiently distinguishing of algorithms on several of the smaller SMT instances).

We presume that A_{best} would be the algorithm delivered to the 'client', to be used to solve future real-world instances. Each of these phases of experimentation (to find A_{best} on a master instance space) typically involved from 4 to 400 billion evaluations: applying c. 200 algorithms to c. 1000 instances, for (usually) 20 trials of between 1,000 and 100,000 solution evaluations each.

Following this, we define several instance subspaces by partitioning $p(I)$ (details later), and we find the A^i_{best} for each subspace i; we then visualise the footprints of chosen algorithms - in particular we view the footprint of A_{best}, typically observing that it covers few of the instance subspaces. Further, when we explore subspaces, we include several outside the master space; this allows us to observe how A_{best} generalises beyond its deemed area of prowess.

The footprint graphics are obtained as follows. Each instance subspace is represented by a specific square. A given algorithm variant is identified by circles of a given shade. E.g. A_{best} (on the master space in context) is always black. If a square contains a shaded circle, then the corresponding algorithm was best in the corresponding subspace. The size of the circle is a measure of signifiance. Following the 'instance subspace' experiments, 1000 randomisation tests are done, in each of which the (best-fitness, evaluation number) pairs for each trial are randomly permuted within the results for the same instance. The points-assignment system (given above) is repeated each time, and this enables us to find (i) the signifiance of any given algorithm with at least the same number of points as the 'best' on that subspace, and (ii) the significance of the difference in performance between the best and second-best algorithms. The first signifiance value was always 1 (100%) in all of our experiments, confirming that the 'all algorithms were equivalent' hypothesis is easily rejected. The second signifiance value varied widely, indicating a 'distance' in performance between the best and second-best algorithms. The size of the circle relates to this second signifiance measure. Broadly speaking, large indicates that the algorithm was much better than the second-best, and that the difference is likely to be significant. A small circle means there was not a very distinct difference between the best and second best. Our code (for all experiments, for the randomisations, as well as producing the footprint graphics) can be obtained from `http://macs.hw.ac.uk/~dwcorne/pubs.htm`

3.2 Footprints in SMT Instance Space

First, experiments were performed to explore the performance of a collection of algorithm parameterisations on a distribution of instances of the single-machine tardiness (SMT) problem. In the SMT variant used here, T tasks, $t_1, ..., t_T$ need

to be scheduled on a single machine, each task t_i having a processing time p_i and a due date d_i. A candidate solution is a permutation of the tasks, defining an order of processing on the machine. Given that order of processing (and with setup times assumed to be zero) each task has a finish time, and an individual task's tardiness is L_i, which is the maximum of $\{0, T_i - d_i\}$. Fitness (to be minimised) is simply the sum of the L_i.

The first 'master' instance distribution was defined by setting the number of tasks to be fixed at 20, and setting uniform ranges of processing times and due dates from the (integer) intervals [10, 40] and [10, 550] respectively. In the second phase, 40 separate subsets of instance distributions were used. Each subset was defined by a (processing time, due date) pair (p, d), and instances were produced by generating 50% of the tasks with processing times uniformly generated from a sub-range of the processing time distribution - the interval $[p, p+10]$ - and with due dates similarly generated from the interval $[d, d+60]$. Further details are in the caption of Figure 1. In this way, each instance generated within an instance subset is 'included' in the master instance space, with a nontrivial chance of being generated in the master instance distribution, however each such subspace is also clearly a well defined subset. Moreover, such a clustered nature in the due date and processing time distributions is consistent with what we may expect for sets of orders that many machine shops need to handle. E.g. clients tend to prefer deliveries at certain times, or certain products or processes may be particularly popular in certain periods.

The set of algorithms tested comprised 204 variants of hillclimbing (HC) and an evolutionary algorithm (EA). The 204 variants emerged from the rates of four distinct operators: adjacent-swap mutation, any-swap mutation, shift-2 mutation, and shift-3 mutation. The first two need no explanation; shift-n mutation means choosing a gene position i uniformly at random, and moving it to $i+n$ (treating the permutation as circular), while decrementing the positions of genes previously in positions $i+1, ..., i+n$. An algorithm configuration was a vector of rates for the first three operators (the probability of the fourth being 1 minus their sum), ranging through all subsets of $\{0.1, 0.2, ..., 0.7\}^3$ whose components summed to 0.9 or below. The second set of algorithms were simple steady state EAs using binary tournament selection, with population sizes in {10, 20, 30}, and operators defined and allowed to vary precisely as described for HC.

Two SMT master instance spaces are considered here, corresponding to 20-task and 30-task problems. For each instance space, we consider footprints at different evaluation limits (1,000, 10,000 and 100,000). The resulting footprints are in Figure 1. The six footprint graphics each show footprints for a selection of five algorithm variants (each with its own shade of grey). In every case, 'black' denotes the algorithm that earned A_{best} on the master instance space. The four other algorithms are those deemed best most frequently over the subspaces. Where no circle appears, this means that the best algorithm for that subspace was best in few (if any) more subspaces It turns out that HC variants were always better than EA variants in the SMT cases studied here; the best algorithm in each subspace was an HC variant.

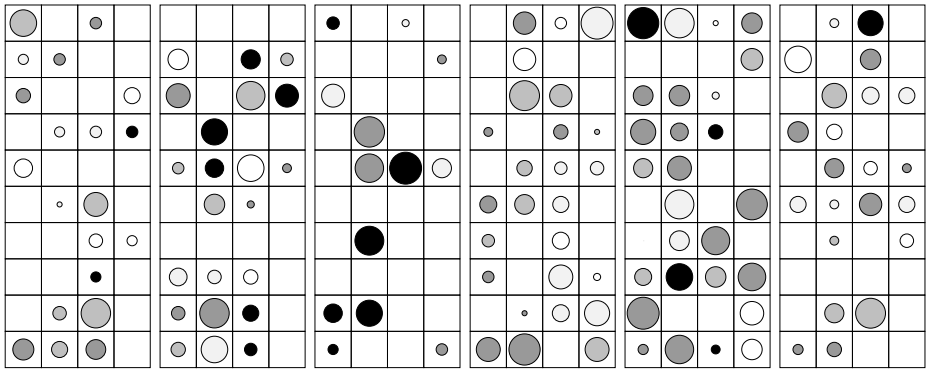

Fig. 1. Six groups of footprints for selected algorithms in SMT instance space. The leftmost (rightmost) three relate to 20 (30)-task problems, and from left to right in each group of three, the footprints relate to max evaluation limits of 1,000, 10,000, and 100,000. Each square indicates an instance subspace, in which the foci for task processing times are, from left to right, [10, 20], [20, 30], [30, 40] and [40, 50], and the due date focus for 20-task problems varies from top to bottom from [10, 70], [70, 130], ..., [550, 610]. For 30-task problems the due dates start at [10,100] and end with [820-910]. The rightmost columns and lowest rows always represent subspaces outwith the master instance space. Black represents A_{best}. Four other footprints are shown in each of the six groups, for the four algorithms that (neglecting the 'black' one) 'won' most frequently among the 40 instance subsets.

Figure 1 reveals a number of interesting points. The 'black' algorithm (always A_{best} for the master space in context) is rarely the best on the subspaces. In practice this suggests that, when an algorithm is tuned on the basis of a master instance space (or perhaps a set of specific instances), the delivered algorithm may rarely be fit for purpose. Footprints are generally patchy, however there are hints that algorithms have 'regions' of prowess. Considering the best in each subspace (including the infrequently winning algorithms in empty cells - whose display in further alternative shades of grey would confuse the visualisation), we explored the level of regionalisation by calculating the degree to which neighbouring (Von Neumann neighbourhood) subspaces shared the same best algorithm. In all cases in Figure 1 this was significant, as evaluated by randomisation tests that permuted 'bests' randomly within the subspaces. Confidence levels were (respectively for the six cases in Figure 1 in left to right order), 98.9%, 90.8%, 99.5%, 98.8%, 99.6 99.6%. The hints of regionalisation in the figures remain clear in the footprints not displayed - although such algorithms tended to win infrequently, their regions of high performance tended to be in the same neighbourhood. Results for subspaces in the rightmost columns and lowest rows speak to generalisation of the 'black' algorithm slightly beyond master instance space. Clearly, the performance of A_{best} beyond the master space is no better than its surprisingly poor showing in subspaces of the master space.

3.3 Footprints in VRP Instance Space

We considered a space of vehicle routing problems (VRP) in which up to three vehicle handle a total of 30 orders, each with a given 60 unit time window. We invented 50 customer locations (and one depot location) uniformly at random from $\{0, 1, 2, ..., 100\}^2$, and a random selection of 10 customers were stamped as 'regulars'. Distances were asymmetric (reflecting some amount of one-way systems and related factors), and generated by adding a uniform perturbation from $[-p, p]$, where p was 20% of the Manhattan distance.

An instance comprised 30 orders, each defined by a customer and a time window. Orders were generated first with a 50% chance of drawing the customer uniformly from regulars, otherwise drawn uniformly from all customers. Then, one of five possible time windows (the first five indicated in the caption of Figure 2) was chosen uniformly at random. A candidate solution was a permutation of 32 numbers, alleles 0–29 indexing the orders, while 30 and 31 were 'dividers', separating vehicles. Reading left to right, from one to three (depending on the divider positions) vehicle routes are calculated in the natural way. Fitness combined distance, lateness and idle time. Full and clear details may be found in the available code.

Instance subspaces were defined by a pair of time windows in which orders would be focussed (e.g. reflecting plausible business practice for many clients). Each subspace corresponded to a pair of the 7 windows listed in Figure 2's caption, as follows: when a customer was (not) regular, there was a 40% chance of the time window being the first (second) of the pair; otherwise time windows were drawn uniformly from all windows.

Figure 2 shows footprints for the cases of maximum evaluations of 10,000 (left) and 20,000 (right). We note first that, in contrast to the SMT cases, the winning algorithms in all subspaces for the VRP were EA variants. Otherwise, much the same can be said as was said for the SMT, although the 'black' algorithm is more prominent in this case. However one clear difference is that fewer distinct algorithms are represented as winners of the subspaces - the mean number of subspaces per winning algorithm was generally c. 3 in the SMT footprints, but c. 4 for the VRP. Also, the appearance of these footprints suggests regionalisation, but in fact the level of neighbour similarity is insignificant in both cases (again deterined by randomisation tests). I.e. there is no more regionalisation than would be expected for a random permutation of the same distribution of winning algorithms within the subspaces. Another difference is that the circles are generally larger. This suggests that choosing an algorithm based on subspace performance in this case may generally be a confident choice, but one which may often generalise poorly, given the general lack of regional similarity.

The nature of the footprints we have observed is clearly a function of the chosen algorithm collection, the problem domains, the instance spaces, and the way we have partitioned instances into subspaces. Further research is needed to understand these dependencies. If, for example there tend to be high levels of regionality for certain algorithms in certain domains, this would allow some confidence for generalisation of performance outside the master space; in other

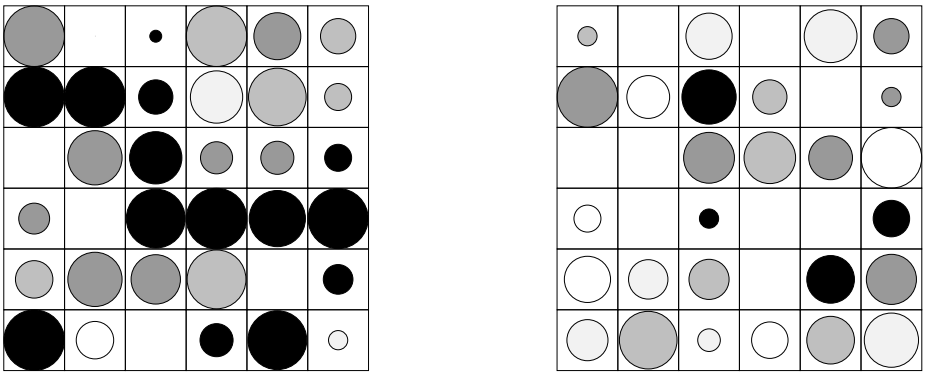

Fig. 2. Two groups of footprints for selected algorithms in a vehicle-routing instance space. Each group relates to a given max number of evaluations: 10,000 (left), 20,000 (right). Instance subspaces are defined by foci on regular and non-regular customer time windows (see text), from left to right (and top to bottom), [60, 120], [120, 180], [180, 240], [300, 360] and [360, 420],[420, 480]. The rightmost columns and lowest rows are outside the 'master' space. Black again represents A_{best} on the 'master' space. Four other footprints are shown, for the (other) algorithms that 'won' most frequently among the 36 subspaces.

cases, it may be that ideal algorithm choice is highly sensitive to the co-ordinates of instance subspace - such a situation demands extra care in delivering the right algorithm, perhaps indicating a per-instance tuning approach. In particular, it could be that distinct tyes of algorithm have distinct footprint characteristics that tend to appear across domains. Especially when prior tuning approaches are costly, such characteristics may be used to inform algorithm choice.

Meanwhlle, the footprints we have examined here clearly challenge certain prior expectations of the robustness of a standard algorithm tuning approach. Our experiments arise from essentially arbitrary but representative and relevant choices of algorithms, domains and instance distributions, and we see similar findings in other domains (work in progress), so we do not expect these to be pathologic cases. Finally, however, we do not claim that the footprints captured herein have clear statistical signifiance in context, but we appeal to their being indicative of the general nature of algorithm footprints in optimisation.

4 Summary and Conclusion

We examined algorithm 'footprints' in the domains SMT and VRP. The footprint of an algorithm indicates how its performance generalises across different dimensions of instance space. In particular, we have found much evidence in support of the claim that the typical way of choosing the 'best' algorithm, via tests over a distribution of instances, is seriously flawed. The algorithm best overall on a broadly defined collection of instances may rarely be best on many well-defined

subsets within that instance space, and similarly outside that instance space. The results also hint at potentially systematic differences between footprints, depending on algorithm family and domain. Such differences, given further understanding in future, may be usefully informative as regards algorithm-choice decisions in many scenarios. In particular, if the natures of footprints tend to generalise well across domains, footprint-oriented algorithm choice may be informative without the need for time-consuming development work that may otherwise be needed.

Acknowledgments. We are grateful for insightful anonymous reviews.

References

1. Hoos, H.H., Stutzle, T. : Stochastic Local Search Foundations & Applications. Morgan Kaufmann (2005)
2. Hutter, F., Hamadi, Y., Leyton-Brown, K., Hoos, H.H. : Performance prediction and automated tuning [...]. CP-06, 213-228. (2006)
3. Hutter, F., Hoos, H.H., Stutzle, T. : Automatic algorithm configuration based on local search, Proc. National Conf. on AI, vol 2, pp. 1152-1157, MIT Press (2007)
4. Hutter, F., Hoos,H.H., Leyton-Brown, K., Stutzle, T. : ParamILS: An Automatic Algorithm Configuration Framework, JAIR, 36, pp. 267-306. (2009)
5. Gratch, J., Chien, S.A. : Adaptive problem-solving for large-scale scheduling problems: A case study. JAIR, 4, 365-396 (1996)
6. Minton, S. : Automatically configuring constraint satisfaction programs: A case study. Constraints 1(1):1-40 (1996)
7. Johnson, D.S. : A theoreticians guide to the experimental analysis of algorithms. Data Structures, Near Neighbor Searches, and Methodology: Fifth and Sixth DIMACS Implementation Challenges, (pp. 215-250). AMS,(2002)
8. Birattari, M.: The Problem of Tuning Metaheuristics as Seen from a Machine Learning Perspective. PhD thesis, ULB, Belgium. (2004)
9. Bartz-Beielstein, T., Lasarczyk, C., Preuss, M. : Sequential parameter optimization, IEEE CEC 2005, pp. 773-780 (2005)
10. Nannen, V., Eiben, A.E. : A Method for Parameter Calibration and Relevance Estimation in Evolutionary Algorithms. GECCO, pp. 183190, New York. ACM.(2006)
11. Nannen, V., Eiben, A.E.: Relevance Estimation and Value Calibration of Evolutionary Algorithm Parameters, IJCAI-97, pp. 975-980(2007)
12. Horvitz, E., Ruan, Y., Gomes, C.P., Kautz, H., Selman, B., Chickering, D.M. : A Bayesian approach [...] . UAI-01, (pp. 235-244). Morgan Kaufmann, (2001)
13. Patterson, D.J., Kautz, H. :(. Auto-WalkSAT: a self-tuning implementation of WalkSAT, Electronic Notes in Discrete Mathematics (ENDM), 9 (2001)
14. Carchrae, T., Beck, J.C. :. Applying machine learning to low-knowledge control of optimization algorithms. Computational Intelligence, 21(4), 372387 (2005)
15. Xu, L., Hutter, F., Hoos, H.H., Leyton-Brown, K. : SATzilla: portfolio-based algorithm selection for SAT. JAIR, 32, pp. 565-60 (2008)
16. Preuss, M. : Adaptability of ALgorithms for Real-Valued Parameter Optimization, in Evoworkshops 2009: 665-674, LNCS (2009)
17. Smith-Miles, K., James, R.J., Giffin, J., Tu, Y. : Understanding the Relationship between [Structure and Performance], in Proc. LION (2009).
18. Edgington, E.S. : Randomization Tests, Marcel Dekker AG, USA, 147pp (1995).