Evaluating the influence of parameter setup on the performance of heuristics for the graph colouring problem

Paulo Neis

Universidade Tecnológica Federal do Paraná (UTFPR),
Curitiba, PR, Brazil
Tel: +55 45 3027 7403 E-mail: neis@neis.com.br

Rhyd Lewis

School of Mathematics,
Cardiff University,
Cardiff, CF24 4AG, WALES
Tel: +44(0)29 208 74856 E-mail: LewisR9@cf.ac.uk

Abstract: This paper aims to analyse the influence of parameter setup over a set of five heuristic methods applied to the graph colouring problem. Each heuristic is applied to a considerable set of problem instances, using a range of different parameter values. Multidimensional analysis is applied to extract and express knowledge about the performance of heuristic methods according to problem instance feature values, highlighting the effect of different parameter setups. The dynamic behaviour of the heuristics is also evaluated at different stages of execution (runtime), providing additional knowledge about speed of convergence/stagnation. Results demonstrate that it is possible to associate regions of the instance space in which problem instances exhibit particular features with specific parameter values yielding superior performance. Information relating runtime with average rate of solution improvement also suggests that certain instance features can be used to determine for how long the heuristics need to run before they converge or stagnate.

Keywords: Parameter tuning; Algorithm performance; Heuristics; Graph colouring.

1 Introduction

The idea of choosing the best algorithm for solving a given problem (or a particular problem instance) before actually solving it has been around for some time. The work of Rice (1976) has laid down the theoretical framework for algorithm selection. This abstract framework, known as “the Rice Model”, describes the mapping between a collection $\mathcal{P}$ of problems (or problem instances) and a collection $\mathcal{A}$ of candidate algorithms applicable to the problems in $\mathcal{P}$. The mapping is established based on a function S that, for a given problem in $\mathcal{P}$, establishes the best algorithm in $\mathcal{A}$, according to some performance criteria and based on a set of features $\mathcal{F}$ of the problems of $\mathcal{P}$. Although appealing, this idea is purely abstract and Rice himself was sceptical that it could lead directly to a superior selection procedure. Recently, in the work of Smith-Miles (2009), this idea has been revisited and actually put into practise for combinatorial optimisation problems (Smith-Miles et al, 2014).

The *No Free Lunch Theorems* (NFL) proposed by Wolpert and Macready (1997) suggest that there is no approach with superior performance in all problem classes. Although the hypothesis of uniformly distributed objective functions assumed by the NFL probably will not hold for a limited number of instances of one particular problem, it has been observed that a set of algorithms rank differently on different problem instances (Lewis et al, 2012; Smith-Miles et al, 2014), therefore suggesting that each algorithm may have strengths or weaknesses in certain subclasses of instances. As pointed out by Rice (1976), problem features can be viewed as a way to introduce the concept of subclasses into the selection model. In this context, the key to the problem characterisation is to determine a set of relevant features, which can be obtained without excessive computational effort, and relate them to expected algorithm performance.

Despite the challenge that algorithm selection represents by itself, finding the most effective set of parameters for a given algorithm could be considered as an algorithm selection problem on its own. Hence in this work we use the term “method” to define the general steps performed by each heuristic method under consideration, and “algorithm” to a specific combination of a method and its parameter setting.

The very definition of “the best parameters” may depend on various criteria. Suppose a method $M \in \mathcal{M}$, with two possible parameter values $\theta_M^{A_1}$ and $\theta_M^{A_2}$, is applied to a sufficiently large set of problem instances. Suppose further that running M with $\theta_M^{A_1}$ results in better solution performance for a certain subset of the problem instances, while runs using $\theta_M^{A_2}$ perform better in a slightly different subset of instances (possibly including some ties). A naive approach would be to declare the “winner” as the parameter that achieves the best performance more frequently, ruling out the alternative. But what if the two combined runs of M perform better than each individual one? In this case, $\theta_M^{A_2}$ may be more appropriate for instances where $\theta_M^{A_1}$ fails, and vice versa. In this case it would be interesting to know whether it is possible to obtain the combined performance of $\theta_M^{A_1}$ and $\theta_M^{A_2}$ without having to execute the method twice. To answer this question it is necessary to know in which instances M produces better solutions using parameter value $\theta_M^{A_1}$, and in which ones $\theta_M^{A_2}$ works better. If the variants of M using the two different parameter values are considered as different algorithms (e.g., A_1 and A_2), therefore composing a new set $\mathcal{A}$ of methods and specific parameter values, then this problem fits the exact definition of the “Algorithm Selection Problem” over the set $\mathcal{A} = \{A_1, A_2\}$.

The focus of this work is on *stochastic search* heuristic methods based on metaheuristics, applied to the graph colouring problem. These are methods in which randomisation plays a prominent role in generating or selecting candidate solutions for a given optimisation problem instance (Hoos and Stützle, 2005). Greedy constructive and iterative improvement procedures are the base of most of these methods. They usually terminate when a complete solution has been generated, a local optimum of a given evaluation function is reached or a computational limit has been exceeded.

This paper aims to discuss three important issues regarding the influence of parameter setups on algorithm performance: (i) Is it possible to identify a subset of instance features strongly influencing the heuristic methods’ performance when different parameter configurations are considered? (ii) Is there a way of linking certain regions of the instance space with specific parameter values of heuristic methods yielding superior performance?

(iii) Is it possible, based on instance features, to determine at what point in the execution the incumbent solutions[1] will stop improving (i.e. the optimum has been found, or the search has stagnated)?

The main contributions of this work are:

- We have applied multidimensional database analysis concepts and tools to extract, visualise and compare information about algorithm performance according to the dimensions (features) of the instance space.
- We have systematically evaluated the influence of parameter setup on the behaviour of heuristic methods. Features of the problem instances have been taken into account for determining the appropriate parameters, producing evidence that an instance-wise tuning of heuristics can lead to improved performance.
- We have investigated the runtime behaviour of algorithms according to features of the instance space, making an assessment of the necessary time to converge or stagnate.
- We have compiled a database about performance of heuristic methods along several dimensions, which includes: problem instance features, resource usage and parameter combination. That information could be further used to select and tune the methods.

The remainder of this paper is organised as follows. Section 2 presents the related work. In Section 3, we describe the methodology adopted in this paper. Section 4 presents the experiments and algorithm setup. Results are presented and discussed in Section 5, with Section 6 concluding the paper and discussing directions for future work.

2 Related Work

This section briefly discusses existing research related to algorithm selection and heuristics parameter tuning. Although this current work should not be considered a study of parameter tuning or an application of algorithm selection, it is closely related to both areas, since it provides a thorough comparison of performance for a set of heuristic methods and a combination of different parameters.

One of the first experimental results establishing a relation between characteristics of instances, search-space and algorithm performance was conducted by Stützle and Fernandes (2004), using the Quadratic Allocation Problem (QAP). In that article the authors evaluated different heuristic methods in a series of QAP instances. Smith-Miles (2008) has extended the results of Stützle and Fernandes (2004) proposing a platform inspired by metalearning to analyse the methods applied to the same optimisation problem. That platform has been applied to a collection of QAP instances using instance features (size of arrays, sparsity, etc.) as the inputs of a supervised learning system with two possible goals: performance prediction or selection of the most suitable method for the problem instance.

Performance assessment of various heuristics methods for the graph colouring problem (GCP) has been tackled by Lewis et al (2012). That paper presented a broad comparison over a considerable set of instances with different features. Later Smith-Miles et al (2014) re-evaluated the same methods over an even larger set of GCP instances using a metalearning approach. The authors concluded that a bespoke hybrid evolutionary algorithm (HEA) proposed by Galinier and Hao (1999) has the best average performance. However, they also demonstrated that in certain regions of the instance space HEA is definitely not the best algorithm.

None of these works, however, conducted a specific analysis regarding the effect of parameter tuning on the heuristic methods' performance. Indeed the comparisons of different methods and the actual selection rarely mentions the parameter configuration being used, leaving a gap to be filled. In essence, parameter tuning consists of selecting the best configuration of an algorithm (Birattari, 2009). Eiben and Smit (2011) have formalised the parameter tuning problem and given an overview of existing parameter tuning strategies for evolutionary algorithms. Different strategies can be classified as on-line and off-line approaches (Birattari, 2009; Eiben and Smit, 2011; Talbi, 2009).

The main idea behind on-line tuning is to modify some parameters of the search algorithm while performing the search itself. In other words, the parameters are controlled and dynamically or adaptively updated during the execution of the heuristic method. A recent work presented by Andersson et al (2016) discusses the advantages of using different parameter sets in different stages of optimisation.

In off-line parameter tuning, the values of different parameters are determined before the execution of the heuristic method (Talbi, 2009). According to Birattari (2009), in most cases heuristics are tuned off-line, by hand in a trial-and-error procedure. Factorial design (which consists in defining the factors to consider and then understanding the relative importance of each parameter) and response surface (described as an iterative search in the space of the parameters (Birattari, 2009)) are other off-line tuning examples. The main disadvantage of the latter is the dependence on a distance metric between each pair of configurations of the heuristic to be tuned (which does not hold in the case of categorical parameters).

Off-line tuning is usually performed one parameter at a time, and in this case no interaction between parameters is taken into account, providing no guarantees that an optimal setting is found (Talbi, 2009). To overcome this problem, the concept of experimental design (Box et al, 2005) can be used. It consists in defining: (i) factors, i.e. the parameters to vary, and (ii) levels, i.e. the different values each parameter can take. In the case of an experiment with j factors in which each factor has k levels, a full factorial design needs k^j executions. Then, the "best" level is identified for each factor. The main drawback of experimental design is the high computational cost, especially when the number of parameters and different values are large, meaning a large number of executions must be performed.

Most works regarding the above strategies do not consider the possibility of tuning parameters according to problem instance features. In the work of Hutter et al (2013) a model-based approach is applied for identifying a set of algorithm parameters and instance features capable of predicting algorithm performance. The approach is then applied to a set of mixed integer solvers and SAT heuristics using a high dimensional parameter space. The results suggest that tuning only a few of the several available parameters, according to certain instance features, is enough to achieve good algorithm performance.

In this paper an experimental design is employed, using a small set of different factors for different heuristic methods, with a maximum of 5 levels for some of the considered parameters. The experiments are conducted with a large number of problem instances exhibiting different features, aiming to identify relationships between instance features and the performance of different parameter setups.

3 Methodology

As mentioned earlier, this work deals with heuristic methods applied to the graph colouring problem. Generally speaking, the adopted methodology consists of:

1. producing a significantly large set of problem instances $\mathcal{P}$, composed of instances with considerable dissimilarity, and calculating each instance's features $F \in \mathcal{F}$;
2. determining the whole set of candidate methods $\mathcal{M}$ and configuration parameters Θ to be evaluated, therefore defining the set of candidate algorithms $\mathcal{A}$;
3. applying each algorithm in $\mathcal{A}$ to each instance in $\mathcal{P}$, keeping track of the results using measures of solution quality and computational resource usage;
4. evaluating the performance of each $A \in \mathcal{A}$ according to the instance features $\mathcal{F}$ and its parameter values $\boldsymbol{\theta}_M^A$ with regard to solution quality and computational effort.

For the set of problem instances used in this study a considerable level of variation is enforced by sampling the instance space, using instances obtained from different sources and generated by different methods. The resulting instance set exhibits a wide range of feature values, spanning a wide region in the instance space (see Section 4.1 for more details).

A set of heuristic methods for tackling with the GCP is employed, along with a grid search on a reasonable range of different parameter configurations for each method. The evaluated parameter values are intended to span a reasonably wide range in the parameter space, capable of bringing forth diverse behaviour of each method in the instance space. At the same time, the number of possible combinations of different parameter values needs to be kept under control, since it can excessively increase the number of executions, rendering the experiments unfeasible due to the resource demand. Performance of the heuristic methods applied to the GCP is measured with respect to the best known solution for each graph G in $\mathcal{P}$.

4 Experiments' Description and Setup

The experiments conducted in this work consist in applying the methodology described in Section 3. First, a large set of instances $\mathcal{P}$ of a particular combinatorial optimisation problem are obtained, the correspondent features are calculated and stored in a database. After that, a collection of well established heuristic methods M is chosen, along with the appropriate parameters $\boldsymbol{\theta}_M^A$, composing the set of candidate algorithms $\mathcal{A}$. Each algorithm in $\mathcal{A}$ is then used to process all the instances in $\mathcal{P}$, collecting performance measures for every single execution. Finally, the resulting database can be processed and analysed in order to extract statistical information about the algorithm's performance.

4.1 Problem and problem instances

The graph colouring optimisation problem can be stated as (Lewis, 2015): given a graph $G = (V, E)$ consisting of sets V and E of n vertexes and m edges respectively, assign each vertex $v \in V$ an integer $c(v) \in \{1, 2, \ldots, k\}$ representing a specific colour such that: (a) $c(v) \neq c(u) \forall \{u, v\} \in E$; and (b) the number of colours k is minimal.

Table 1 Graph colouring problem instances.

Family	Source	Number of instances
C1	Culberson's Generator (Culberson, 2002)	1,000
C2	From authors of (Smith-Miles et al, 2014)	965
C3	Culberson's Generator	1,000
C4	Culberson's Generator	1,000
C5	From authors of (Smith-Miles et al, 2014)	1,000
D	DIMACS competition and Networkx Generator	792
E	From authors of (Lewis et al, 2012)	20
F	From authors of (Lewis et al, 2012)	80
G	From authors of (Lewis et al, 2012)	13
H	Culberson's Generator	103
I	Culberson's Generator	57
Total:		6030

Given an instance of this problem, a candidate solution that assigns colours to all vertexes V is called "complete", the opposite being a "partial" solution. A candidate solution that assigns different colours to every pair of vertexes joined by an edge (also said to contain no clashes) is called "proper", the opposite being an "improper" solution. In order to be feasible, the candidate solution needs to be both complete and proper. The problem statement requires the optimal solution to be feasible and to use the smallest possible number of colours $\chi(G)$, called the graph's "chromatic number".

In terms of computational complexity the general form of the GCP is considered to be intractable (NP-hard), with its decision variant belonging to the class of the NP-complete problems. Therefore in practise approximation algorithms and heuristics are used to obtain acceptable, possibly sub-optimal solutions. These compromise solutions may not reach the true minimum number of colours $\chi(G)$ for a given instance, but they are obtained using a reasonable amount of computational resources. More importantly, they are valuable for the many practical applications based on the GCP (Hardy et al, 2017; Lewis, 2015).

The set of problem instances used in this experiment is composed of 6030 graphs[2]. The instances are grouped into subsets or "families" of instances, according to the source from which they were obtained or the method employed to generate them[3]. A summary of the instances is provided in Table 1.

4.2 Heuristic methods and candidate algorithms

The set of methods $\mathcal{M}$ evaluated in this work is a subset of those described by Lewis (2015)[4], particularly the ones exhibiting externally configurable parameters. This subset consists of the following methods:

- HillClimber (HC): The hill climbing strategy proposed by Lewis (2009) for grouping problems. HC operates on a single feasible candidate solution using a constructive heuristic, and iteratively improving it by applying a specialised local search (LS) operator. At the beginning of each cycle, the method takes the initial proper solution $\mathcal{S} = \{S_1, S_2, \ldots, S_{|\mathcal{S}|}\}$, where $\mathcal{S}$ is a partition of the vertex set, and each $S_i \in \mathcal{S}$ is an independent set. It then removes a small number of these independent sets (colour classes) and places them into a second set $\mathcal{T}$, resulting in two partial proper solutions.

The LS operator is then applied for I_{HC} iterations, attempting to transfer vertexes from colour classes in $\mathcal{T}$ into colour classes in $\mathcal{S}$, such that both $\mathcal{S}$ and $\mathcal{T}$ remain proper, thus increasing the cardinality of the classes in $\mathcal{S}$ and possibly emptying some classes in $\mathcal{T}$, reducing the total number of colours. At the end of the cycle, all remaining non-empty classes in $\mathcal{T}$ are moved back into $\mathcal{S}$, and its independent sets are re-ordered and modified using greedy heuristics before the LS operator is repeated. I_{HC} is the parameter available for configuration, and assumes integer values. The experiment has been conducted varying the parameter around the default value configured in the source code (Lewis, 2015), as described in Table 2.

- TabuCol: A local search heuristic implementing a simple tabu search paradigm applied to the graph colouring problem (Galinier and Hao, 1999). TabuCol operates in the space of complete improper solutions (solutions that assign colours to all vertexes of the graph, possibly containing clashes), defining a cost function that is proportional to the number of clashes produced. This method is also used as local search subroutine in some of the hybrid heuristics listed below. TabuCol has a categorical parameter called the *"tenure type"*, simply referred to as T_{TC}. The tabu tenure t, which is the number of iterations that a particular vertex is forbidden from being reassigned to a particular colour, is determined according to T_{TC}. There are two possible strategies: (i) *"dynamic"*, which consists of making $t = UNIFORM(a, b) + \alpha \cdot C$, with $a = 0$, $b = 9$ and $\alpha = 0.6$ (in other words, making t a random variable proportional to the incumbent solution's cost C); or (ii) *"reactive"* (default), which consists in tuning[5] the tenure based on the variation Δ between the maximum and minimum values of the objective function calculated every ϕ iterations, and making $t = t + \eta$ (if $\Delta \leq 1$ or $t = 0$) or $t = t - 1$ otherwise[6] (Blöchliger and Zufferey, 2008).

- PartialCol: the scheme proposed by Blöchliger and Zufferey (2008) is also based on tabu search, but considering only feasible partial solutions. That is, clashes are forbidden but some vertexes may remain uncoloured. The same categorical parameter used in the TabuCol method is available here, being referred to as T_{PC}, and also takes either of the *"reactive"* (default) or *"dynamic"* strategies.

- AntCol: is a population-based method inspired by the *Ant Colony* metaheuristic that has been enhanced by Dowsland and Thompson (2008) for the GCP. At the completion of each cycle, the TabuCol heuristic is applied to each candidate solution. The parameter available for configuration is referred to as I_{AC}, and controls the number of iterations of the tabu search per cycle, assuming integer values. The default setup is $I_{AC} = 2 \times |V|$ iterations, thus being proportional to a graph's order.

- HEA: the Hybrid Evolutionary Algorithm for the GCP was originally proposed by Galinier and Hao (1999), and is considered to be one of the best performing methods for this problem (Lewis et al, 2012; Smith-Miles et al, 2014). HEA operates on a population of candidate solutions which are evolved using a recombination (crossover) operator, interlaced with a local (tabu) search procedure. Being a more sophisticated procedure, HEA has a larger set of control parameters. In this work, we evaluate three of them: (i) P_{HE}, the *"Population Size"*, which is an integer parameter assuming a default of 10 individuals; (ii) I_{HE}, an integer parameter that controls the number of iterations of the tabu search per cycle, being the parameter value internally multiplied by the graph's order $|V|$ to obtain the actual iteration limit; and (iii) X_{HE}, the *"Crossover Type"*, a categorical parameter that controls how the recombination of candidate solutions

Table 2 Methods and parameters used.

Method (M)	**Parameters**	**Values** (Θ_M)	**Comb.**
AntCol	I_{AC} (iteration limit)	{1, **2**, 16, 32, 64}	5
TabuCol	T_{TC} (tenure type)	{Dynamic, **Reactive**}	2
PartialCol	T_{PC} (tenure type)	{Dynamic, **Reactive**}	2
HillClimber	I_{HC} (iteration limit)	{100, 500, **1000**, 5000, 10000}	5
	I_{HE} (iteration limit)	{1, 8, **16**, 32, 64}	
HybridEA	P_{HE} (population size)	{4, **10**, 40}	30
	X_{HE} (crossover type)	{**GPX**, GPX+Kempe}	
Total number of resulting algorithms:			44

(parents) into new candidate solutions (offspring) is carried out. Two different options of X_{HE} have been experimented with: the default *Greedy Partition Crossover* ($X_{HE} = GPX$) of Galinier and Hao (1999), which constructs offspring using large colour classes from both parents, not necessarily leading to proper solutions; and a refined version of GPX described by Lewis (2015), which first removes any clashing vertexes from each parent, making the offspring proper, before altering this proper solution by means of *Kempe chain* interchanges.

Table 2 presents the sets of parameters used with each heuristic method $M \in \mathcal{M}$. The default values used in the source code are shown in bold. The combination of the five methods with every respective parameter values results in a set $\mathcal{A}$ with $|\mathcal{A}|$= 44 different algorithms. Although we acknowledge the existence of other high performance heuristic methods for the GCP (Aranha et al, 2018; Hertz et al, 2008; Malaguti et al, 2008; Moalic and Gondran, 2017; Porumbel et al, 2010; Titiloye and Crispin, 2011; Wu and Hao, 2012), our analysis is restricted to the algorithms implemented in the particular software package we used. Moreover, we emphasise that the objective of this study is not to benchmark heuristics. We also acknowledge that other parameters inherent to each of these heuristic methods could be explored, however that would involve making changes to the software package, which is beyond the scope of this study. Given the objectives of this work is not to establish a rank of best performing heuristics, the insights obtained by studying this subset of methods can provide guidelines for extending the analysis to other heuristics and even other problems.

4.3 Measuring performance

The performance metrics, with respect to solution quality, are evaluated relative to an upper bound on $\chi(G)$, denoted by $\overline{\chi}(G)$. This upper bound consists in the solution with fewer colours among the set of all solutions produced by the algorithms in $\mathcal{A}$ for each graph G We consider three different metrics of solution quality: (i) The frequency (or proportion) with which a given algorithm reaches $\overline{\chi}(G)$ for instances in a given subset of $\mathcal{P}$; (ii) the frequency with which a given algorithm fails to produce proper solutions for instances in a given subset of $\mathcal{P}$; (iii) a measure of *relative deviation of solution quality* (Hoos and Stützle, 2005). The latter can be interpreted as a "distance" $D_B(G)$, proportional to the difference between the number of colours $\chi_c(G)$ of a given candidate solution for graph G and $\overline{\chi}(G)$, defined by Eq. (1):

$$D_B = \frac{\chi_c(G) - \overline{\chi}(G)}{\overline{\chi}(G)} \tag{1}$$

The concept of "distance" can be used to compare the performance of algorithms on graphs with very different chromatic numbers, for instance: Consider two graphs G_1 with $\chi(G_1) = 5$ and G_2 with $\chi(G_2) = 100$, and two respective candidate solutions having $\chi_c(G_1) = 6$ and $\chi_c(G_2) = 101$ colours. They are both off by one colour from the global optimum, however it seems reasonable to assume that $\chi_c(G_2)$ is relatively "closer" to the global optimum than $\chi_c(G_1)$, considering the chromatic number of each graph. Performance measures are required to be in the same "base", so results can be aggregated over a large number of varied problem instances and heuristic methods. Defining D_B values in this way make it independent of graph size and chromatic number, facilitating direct comparison and averaging of results over any subset of $\mathcal{P}$. Another advantage of this measure is the possibility of using it to study the dynamic behaviour of algorithms, such as stagnation during execution.

4.4 Data analysis and performance comparison

For our experiments, results are organised into a database containing: problem instance data (name, features), algorithm data (M,Θ_M) and execution data (number of colours, computational effort and time spent), along with some other derived information. The analysis is performed using concepts of multidimensional database analysis (Pedersen and Jensen, 2001), i.e., we associate dimensions in the database with performance measures. Continuous numerical features are discretized into ranges in order to define dimensions, along with categorical data. Dimensions are used for selecting and aggregating results at the desired level of detail, for example: taking a subset of $\mathcal{P}$ based on certain feature values and averaging the solution distance metric.

The features chosen for problem instance characterisation are among the ones described in the work of Smith-Miles et al (2014). Initially 14 features were calculated for the problem instances. Those features were then filtered using the concept of information gain (IG) (Liu and Motoda, 1998) based on the Shannon's entropy measure, in order to determine the most "influential" ones. To apply this concept, each problem instance/algorithm pair is considered as a data point, and a quality measure of the solution produced by that algorithm at the end of its execution cycle is discretized into one of the three following classes: (i) solutions whose $D_B = 0$ are attributed to the class "*best*"; (ii) solutions whose D_B is less or equal to average are attributed to class "*good*"; (iii) the remaining are attributed to class "*poor*". By calculating IG with respect to this classification for each of the 14 features, and ranking them, the first two more relevant were chosen (1 and 2 listed below), in order to restrict the dimensions in the analysis and allow us to visualise 2D plots of the instance space. After further interactive analysis of the database, it was verified that the metric of algorithm failures is strongly influenced by a third feature (3 listed below), ranked low in the measure of IG. This third feature is included in some of the analysis performed in the following sections. Therefore, the set of problem instance features is composed of:

1. Standard deviation of betweenness centrality σ_{CB}: Betweenness centrality $C_B(v)$ offers a measure of how central a given vertex v is to the graph, being defined as the

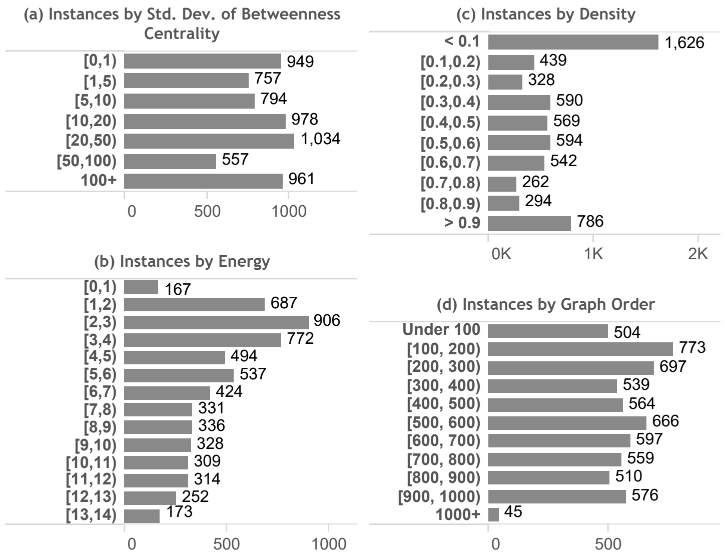

Figure 1 Distribution of instances by feature value.

number of shortest paths that pass through v. Therefore the standard deviation of this measure over the whole set V represents the dispersion of this property within the graph.

2. Energy $\mathcal{E}(G)$: The term "energy" has its origins in theoretical chemistry, and is defined as the mean of the absolute values of the eigenvalues of the adjacency matrix[7].

3. Graph density $\rho(G)$: This is defined as the ratio of the number of edges to the number of possible edges ($\rho = 2m/n(n-1)$).

Fig. 1 depicts the distribution of the problem instances used in our experiment, according to each of these features. The graph order is also included to provide a more intuitive measure of "instance size".

To compare the performance of different algorithms in $\mathcal{A}$ on subsets of $\mathcal{P}$, statistical significance tests are applied with a significance level of $\alpha = 0.01$. If the measure being compared is the proportion of best known solutions found by two algorithms, the chi-square test for differences among proportions is applied (Levine et al, 2007). This is the case when two algorithms are compared according to the number of times each one has found $\overline{\chi}(G)$. If more than two algorithms are being compared, a pairwise comparison is performed with Holm-Bonferroni correction for multiple hypothesis testing. If the measure being compared is the solution distance, the Friedman test (Demšar, 2006) is applied. For cases where more than two algorithms are compared among themselves, the Friedman test is initially applied to determine if the null hypothesis can be rejected (at least one of the algorithms obtains significantly distinct results). Given that the null hypothesis is rejected, a post-hoc Nemenyi test is applied to determine which algorithms have significantly superior performance.

The amount of resources needed to achieve a certain performance (averaged for a subset of instances with similar features) can be an important factor in determining the most appropriate algorithm to be used with a given instance. As a machine-independent measure of resource usage and an attempt to perform a fair comparison, the concept of "constraint check" (Lewis, 2015) is employed by defining the same budget of solution evaluations for all the compared algorithms. This strategy was chosen to give algorithms a fair chance of achieving a good solution while keeping the computational time within acceptable limits.

Summarising, the experiment setup encompasses a set of over six thousand problem instances and 44 algorithms, resulting in more than 260,000 independent executions[8]. The experiments have been conducted using a heterogeneous cluster equipped with "*Intel Core i5*" and "*Intel Core i7*" processors, capable of running a total of 64 parallel and independent threads. Computers in the cluster run the Debian/Linux operating system[9]. Job management is performed through scripts and the GNU Parallel tool[10]. To complete its execution, each algorithm is allowed to run for a maximum of 5×10^{10} constraint checks. This value is a the same employed in the work of Smith-Miles et al (2014), and a compromise compared to those used by Lewis et al (2012), which have employed a limit of 5×10^{11} constraint checks. The constraint checks counter is incremented globally in each individual algorithm's execution, meaning that it never decreases or gets reset, even on hybrid methods and heuristics using "restarts". A hard limit of 20 minutes of processor time is also set for each individual algorithm execution, in order to prevent hung processes from clogging the cluster. Individual executions that violate this hard limit are forcefully terminated. If a particular algorithm does not produce a proper solution for a given problem instance after using up its resource limit it is said to have failed for that instance, and its solution distance is defined as one. Under these conditions, the overall time to complete the experiment in the cluster was about 14 days. Roughly speaking, if the experiment were executed in a single core machine, we estimate that it would have taken about 896 days (approximately $2\frac{1}{2}$ years) to complete.

5 Results

In this section the performance measures proposed in Section 4.3 are presented both overall and along each of the feature values considered for analysis. Multidimensional database analysis concepts and tools are applied in order to extract, visualise and compare information about algorithm performance. A simple method of "slicing" the instance space is applied to determine the best performing algorithm in each region.

5.1 Overall algorithm performance across all problem instances

This section focuses on analysing performance across the entire space of problem instances, whilst the next section analyses performance of algorithms in specific regions of the instance space. A summary of performance metrics for each of the methods using various parameter values is given in Table 3. The number and percentage of best known solutions found are presented, along with the percentage of algorithm failures, and the average and standard deviation of solution distance $D_B(G)$. Only the default, the best and the worst performing parameter configurations for each method are shown. The best results among all the algorithms are shown in bold.

The results in Table 3 emphasise the importance of using a proper parameter setup. For the AntCol and HEA methods there is a considerable difference in performance between the best, default and worst performing parameter configurations. For both these methods we performed a statistical comparison of the two populations composed of the results obtained respectively by the default and the best global performer parameters. The pairwise comparison of proportions on the number of best solutions found in each population lead

Table 3 Overall results: Performance metrics over the whole instance space.

Method	param values	# best	% best	% fail	Avg dist	Std dist
AntCol (I_{AC}=)	1 (worst)	1,408	23.2	33.2	0.3524	0.4584
	2 (default)	1,741	28.7	25.6	0.2771	0.4255
	32 (best)	3,309	54.6	6.8	0.0881	0.2503
TabuCol (T_{TC}=)	Reactive (default/best)	3,682	60.2	3.38	0.0468	0.1807
	Dynamic (worst)	3,525	58.1	4.74	0.0614	0.2110
PartialCol (T_{PC}=)	Reactive (default/best)	3,330	54.9	4.78	0.0637	0.2114
	Dynamic (worst)	3,203	52.8	4.79	0.0662	0.2117
HillClimber (I_{HC}=)	100 (worst)	2,776	45.8	1.72	0.0935	0.1523
	1,000 (default)	2,865	47.2	1.71	0.0866	0.1469
	10,000 (best)	2,925	48.2	1.87	0.0785	0.1477
HEA	{16/10/GPX} (default)	4,301	70.1	1.49	0.0239	0.1221
(I_{HE}/P_{HE}/X_{HE}=)	{8/4/GPX} (best)	**4,780**	**78.8**	**1.39**	**0.0214**	**0.1189**
	{1/4/GPX} (worst)	3,228	53.2	1.56	0.0358	0.1279

to the rejection of the null hypothesis. In other words, there is strong evidence that some parameter configurations perform better than others when the whole set of available problem instances is considered.

For the HillClimber method the performance seems to be affected by the parameter controlling the local search iteration limit, favouring longer iteration cycles. However the pairwise comparison of proportions is inconclusive at the 0.01 significance level.

Regarding PartialCol and TabuCol, the default parameter configuration performed better than the alternative one when the whole instance space is considered, suggesting that the default reactive mechanism proposed by Blöchliger and Zufferey (2008) does bring a benefit in general.

From a global perspective, the HEA method performed better than the other methods, particularly when using a parameter configuration different from the default. Regarding the two crossover operators tested, the default (GPX) has overall performance marginally better than the alternative (GPX + Kempe).

5.2 Performance according to regions of the instance space

In this section a multidimensional analysis is conducted on the experimental data in order to verify the hypothesis that selecting and tuning the heuristic methods according to features of the problem instance can yield superior performance than the overall best results described in Section 5.1.

5.2.1 Proportion of best solutions found

Tables 4 and 5 present the number of best solutions found by each algorithm in "slices" of the instance space according to the dimensions (problem features) σ_{CB} and $\mathcal{E}(G)$ respectively. The best result in each column is highlighted in red, and light grey cells indicate the results that were not significantly different from the best result, based on the pairwise comparison of proportions, relative to the total number of instances in that column.

The distribution of the grey cells along these two dimensions suggest the existence of specific regions of the instance space where parameter tuning may yield superior performance. Analysing the data in Table 4, it seems that the HillClimber method performs

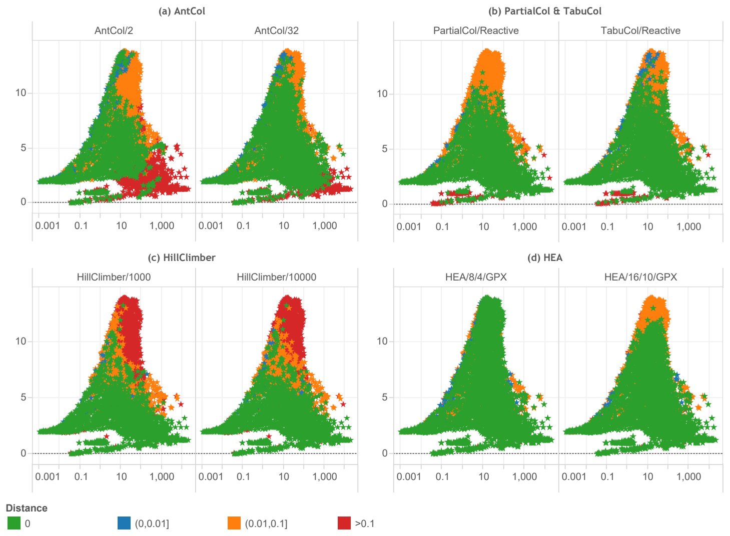

Figure 2 Performance of some algorithms, in terms of solution distance, compared in a two-dimensional space.

well on extremely low values of σ_{CB} (< 1). In this particular range of σ_{CB}, HillClimber seems to be relatively insensitive to the I_{HC} parameter, with a small advantage for $I_{HC} = 100$. Although several parameter configurations of HEA show a performance similar to HillClimber, the former is more sensitive to parameter configuration changes, since some of its configurations (including the best overall performer) are significantly worst than any variant of HillClimber.

Table 4 shows that HEA clearly has an advantage over a wide range of values of the σ_{CB} dimension. For some ranges of this dimension, clear favourable parameter values can be pointed to. However, at higher values of σ_{CB} both TabuCol and HEA show good performance for some parameter values.

In contrast, Table 5 shows that HEA performance becomes more sensitive to parameter tuning as $\mathcal{E}(G)$ increases, with specific parameter values being favoured. This advantage on high energy instances ($\mathcal{E}(G) > 8$) is also clear when comparing HEA to other methods.

5.2.2 Spatial distribution of solution distances

Fig. 2 shows the distribution of problem instances in a two dimensional space determined by the features σ_{CB} (horizontal axis, log scale) and $\mathcal{E}(G)$ (vertical axis, linear scale). In this two dimensional space each problem instance is colour-coded according to the performance of the algorithms to obtain a "footprint" indicating where each one performs best. The colour code represents ranges of solution distances, as defined by Eq. 1. Green colour means that $\overline{\chi}(G)$ has been reached. Only the default and overall best performing parameters for each method are shown.

Fig. 2 (a) compares AntCol's default parameter with its best observed parameter setup. We see that its footprint is extended to a larger portion of the instance space using parameter values different to the default ones. Similarly, HEA footprint is also stretched relatively to the default, as shown in Fig. 2 (d). Indeed the alternative parameter configuration shows

Table 4 Number of best solutions found, broken down by different ranges of σ_{CB}

		σ_{CB}						
Method	Params	[0,1)	[1,5)	[5,10)	[10,20)	[20,50)	[50,100)	[100,+)
	1	647	350	163	100	53	31	64
	2	657	401	222	174	144	62	81
AntCol	16	667	491	415	471	460	296	364
	32	673	509	410	461	449	317	490
	64	669	501	389	392	432	328	584
TabuCol	*Reactive*	692	508	376	360	505	417	824
	Dynamic	599	513	354	346	508	418	787
PartialCol	*Reactive*	631	436	310	312	467	392	782
	Dynamic	604	422	273	282	460	393	769
	100	**838**	461	215	187	238	240	597
	500	826	453	203	202	248	258	638
HC	1000	816	446	208	201	260	271	663
	5000	808	428	209	211	273	291	694
	10000	815	428	202	206	275	291	708
	1/4/GPX	741	464	253	275	377	336	782
	1/4/GPX+	801	498	269	269	397	348	782
	1/10/GPX	755	469	305	373	551	388	791
	1/10/GPX+	797	505	309	361	563	409	796
	1/40/GPX	749	465	331	367	530	411	810
	1/40/GPX+	776	496	341	383	533	388	752
	8/4/GPX	763	572	**587**	**746**	**825**	**468**	819
	8/4/GPX+	823	597	547	669	656	416	806
	8/10/GPX	778	569	499	598	712	456	829
	8/10/GPX+	806	586	509	572	639	401	792
	8/40/GPX	775	524	401	435	559	410	834
	8/40/GPX+	800	540	405	437	524	372	806
	16/4/GPX	772	602	543	646	734	449	824
	16/4/GPX+	818	**615**	484	520	573	396	806
HEA	16/10/GPX	779	575	480	542	657	441	827
	16/10/GPX+	814	586	461	503	566	389	805
	16/40/GPX	790	544	383	417	530	408	832
	16/40/GPX+	802	542	378	393	490	372	811
	32/4/GPX	783	602	481	530	638	430	824
	32/4/GPX+	821	599	427	442	517	386	818
	32/10/GPX	783	573	438	472	598	416	837
	32/10/GPX+	810	587	409	421	501	382	818
	32/40/GPX	792	548	366	377	500	394	842
	32/40/GPX+	810	553	353	358	459	367	834
	64/4/GPX	787	582	436	450	566	417	824
	64/4/GPX+	816	599	399	377	508	391	827
	64/10/GPX	786	561	406	405	537	413	831
	64/10/GPX+	805	566	376	379	491	384	832
	64/40/GPX	795	547	359	357	487	391	**849**
	64/40/GPX+	804	551	342	342	460	380	843
Total Instances		949	757	794	978	1034	557	961

Table 5 Number of best solutions found, broken down by different ranges of $\mathcal{E}(G)$

		$\mathcal{E}(G)$													
Method	Params	[0,1)	[1,2)	[2,3)	[3,4)	[4,5)	[5,6)	[6,7)	[7,8)	[8,9)	[9,10)	[10,11)	[11,12)	[12,13)	[13,14)
AntCol	1	106	264	246	209	171	171	127	43	26	18	9	4	4	10
	2	106	265	303	275	225	234	157	75	32	21	10	7	10	21
	16	106	351	575	547	334	343	245	185	154	113	66	53	50	42
	32	108	408	621	576	342	368	273	206	181	130	64	21	8	3
	64	115	460	650	583	356	385	277	208	154	72	22	8	3	2
TabuCol	*Reactive*	59	595	**875**	641	388	416	306	191	110	56	28	14	3	0
	Dynamic	56	583	853	585	351	390	302	172	97	62	37	20	14	3
PartialCol	*Reactive*	56	580	859	624	349	370	259	131	53	37	10	2	0	0
	Dynamic	56	575	852	601	335	347	239	97	43	34	12	5	4	3
HC	100	159	661	846	557	218	166	83	38	18	16	8	3	1	2
	500	160	659	858	600	243	154	73	35	18	14	9	3	0	2
	1000	158	661	859	613	246	171	76	40	16	13	9	2	0	1
	5000	159	660	853	645	273	176	78	34	13	13	7	2	0	1
	10000	160	656	850	656	285	175	76	33	12	12	7	2	0	1
HEA	1/4/GPX	161	671	854	598	306	286	167	64	31	29	16	7	12	26
	1/4/GPX+	159	667	856	649	339	315	200	81	35	27	13	4	6	13
	1/10/GPX	159	670	851	610	309	319	204	115	66	56	40	62	99	72
	1/10/GPX+	159	673	856	640	335	340	252	125	78	58	42	60	79	43
	1/40/GPX	162	670	855	635	328	329	239	128	104	112	90	11	0	0
	1/40/GPX+	162	671	860	615	334	360	253	133	122	102	57	0	0	0
	8/4/GPX	163	673	858	631	347	377	300	205	206	197	202	**249**	**220**	**152**
	8/4/GPX+	163	673	865	**673**	358	388	314	209	193	168	189	167	94	60
	8/10/GPX	162	**676**	866	643	356	403	320	231	232	**242**	**209**	90	11	0
	8/10/GPX+	162	673	865	654	361	399	315	236	233	218	159	30	0	0
	8/40/GPX	163	673	866	659	361	406	307	214	187	87	14	1	0	0
	8/40/GPX+	161	674	869	654	366	395	294	219	170	69	12	1	0	0
	16/4/GPX	162	674	862	631	357	393	313	230	233	217	185	172	94	47
	16/4/GPX+	161	672	867	668	374	398	317	224	197	148	112	56	15	3
	16/10/GPX	164	674	867	642	364	409	**332**	**239**	**248**	212	120	28	2	0
	16/10/GPX+	163	675	873	657	366	398	322	232	210	155	63	10	0	0
	16/40/GPX	164	675	872	658	379	411	316	217	149	49	11	3	0	0
	16/40/GPX+	163	672	870	658	369	385	304	205	121	28	10	3	0	0
	32/4/GPX	163	671	864	641	369	404	323	233	210	182	134	64	21	9
	32/4/GPX+	163	**676**	871	670	378	410	315	201	138	106	63	15	3	1
	32/10/GPX	**165**	**676**	871	648	376	**417**	327	238	200	137	52	8	1	1
	32/10/GPX+	**165**	673	871	665	373	396	306	210	149	87	25	6	1	1
	32/40/GPX	164	672	873	658	384	404	313	194	108	32	12	3	1	1
	32/40/GPX+	164	675	871	664	381	392	292	169	80	29	12	3	1	1
	64/4/GPX	**165**	675	866	639	374	401	328	217	175	124	66	26	4	2
	64/4/GPX+	164	673	874	**673**	**394**	402	310	199	118	72	27	8	2	1
	64/10/GPX	162	674	869	654	377	407	315	205	150	92	25	7	1	1
	64/10/GPX+	160	673	874	670	389	397	299	187	106	51	20	5	1	1
	64/40/GPX	163	673	872	664	387	410	302	180	80	32	15	5	1	1
	64/40/GPX+	**165**	673	**875**	666	392	392	288	157	63	31	13	5	1	1
Total Instances		167	687	906	772	494	537	424	331	336	328	309	314	252	173

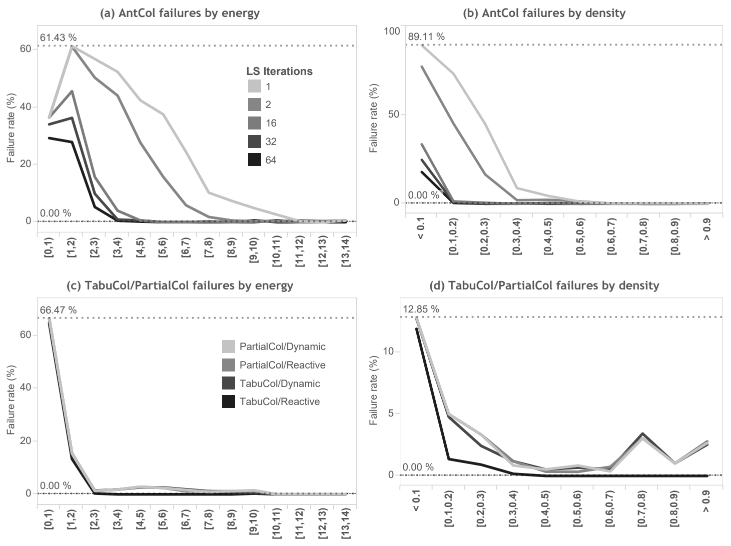

Figure 3 AntCol, TabuCol and PartialCol algorithm failures according to instance properties

a clear advantage in the high $\mathcal{E}(G)$ and medium σ_{CB} region, consistently with the results reported in Tables 4 and 5.

HillClimber on the other hand has no visual improvement, since the gain relative to the default parameter is less than one per cent, according to Table 3. Similarly TabuCol and PartialCol also showed no improvement under the alternative parameter setups, therefore Fig. 2 (b) shows the footprints of both using the default tabu strategy.

5.2.3 Algorithm failures

The algorithm failure rate may be an important performance measure, particularly if it could be linked to problem instances showing specific features. A given algorithm may show high failure rates, and therefore a "weakness", in certain regions of the instance space where some other algorithm may be more successful. Therefore, this measure might also be used as a "tie breaker" in case other performance measures cannot conclusively point to a favourable algorithm. Additionally, identifying particular feature values that lead to a poor performance of a given method provides the algorithm's designer or developer with hints of where to start looking for improvements or bug fixes to his/her design or implementation. In this section only the features $\mathcal{E}(G)$ (energy) and $\rho(G)$ (density) are shown, since they provide a better separation into regions of high failure rates.

Fig. 3 (a) and (b) show the failure rate of AntCol along the dimensions $\mathcal{E}(G)$ and $\rho(G)$, respectively. The data suggests that AntCol has a surprisingly high failure rate with instances of low energy and low density. The reasons for the high failure rate observed with this AntCol implementation requires further investigation.

Fig. 3 (c) and (d) show the failure rate of TabuCol and PartialCol. TabuCol shows high failure rate on problem instances whose $\mathcal{E}(G)$ range is below 2, regardless of T_{TC} value. However, with instances above this threshold the failure rate is very low. Along the $\rho(G)$ dimension the separation is not so good; however it can be seen that both high and low density regions show high failure rate. PartialCol behaves similarly to TabuCol.

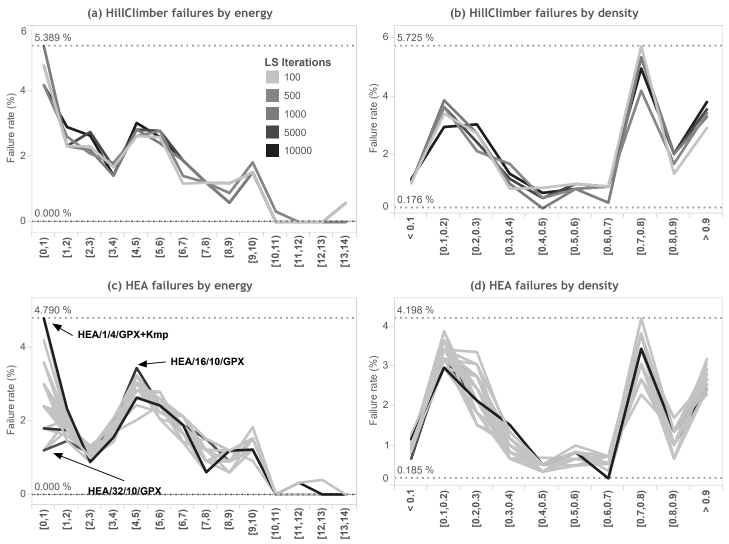

Figure 4 HillClimber and HEA algorithm failures according to instance properties

Fig. 4 compares the failure rate of HEA and HillClimber. These methods show fairly low failure rate of less than 6% in the worst case, regardless of the parameter tuning. Along the $\rho(G)$ dimension the separation is not so clear; however, the extremely low values of $\rho(G)$ that were associated with high failure of the previous three methods seem to present no problems for HEA and HillClimber.

5.2.4 Dynamic behaviour

We now consider the behaviour of the algorithms during execution. Fig. 5 depicts the average solution distance (D_B) according to the number of constraint checks performed. Here the instance space is partitioned into four energy ranges, and the average D_B across all instances in each range is represented. Each line corresponds to one algorithm, and different colours correspond to different heuristic methods. Logarithmic scales are used in both axes to reinforce the differences.

Apparently, for low energy instances ($\mathcal{E}(G) < 2$, Fig. 5 (a)), the algorithms converge quickly with little improvement after the 10^7 constraint checks limit. In particular, AntCol, TabuCol and PartialCol are completely stagnated beyond 10^8 checks, suggesting that they could have been executed using 100 times less resources and yet reach the same results. These three methods show considerably larger average distance in this energy range. This can be traced back to the high failure rate of these algorithms in this energy range, shown in Fig. 3. At the final stage of execution, HEA with the parameter combination $\{I_{HE}$=32,P_{HE}=10,X_{HE}=GPX$\}$ has reached the smallest average distance for instances below the energy threshold of 2, suggesting that this combination has the best average performance for this set of problem instances. The grey band in Fig. 5 (a) shows the interval where the Friedman pairwise comparison was inconclusive at 0.01 significance level. The comparison is performed using the set of distances reached by each algorithm in every problem instance after 5×10^{10} constraint checks. In other words, the performance of HEA using parameter setting $\{I_{HE}$=32,P_{HE}=10,X_{HE}=GPX$\}$ cannot be conclusively distinguished from any other setting of the HEA or HillClimber methods

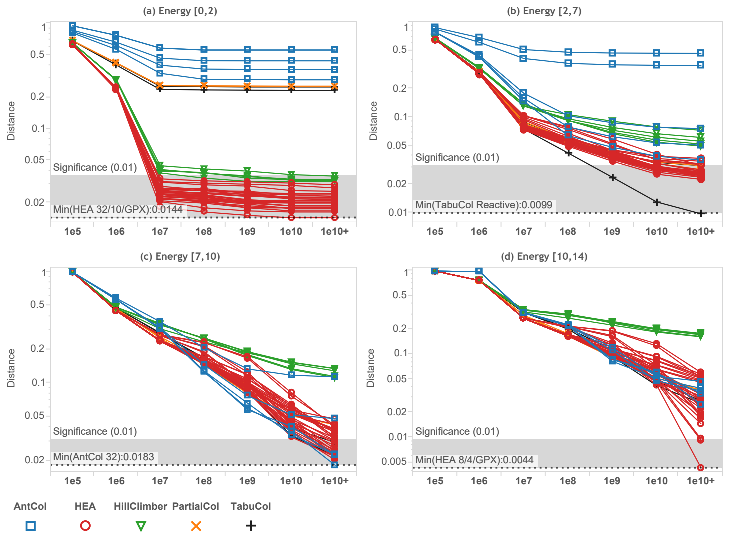

Figure 5 Average solution distance along algorithm execution for different energy ranges

and the corresponding parameters experimented with. However HEA's performance is significantly different from the performance of AntCol, TabuCol and PartialCol, regardless of their parameter tuning.

In the energy range $\mathcal{E}(G) \in [2, 7)$, shown in Fig 5 (b), there are some signs of search stagnation for some algorithms. TabuCol with the default T_{TC}=Reactive tabu strategy reaches the smallest average distance. The grey band represents the region where the Friedman pairwise comparison was inconclusive, suggesting that, for some parameter configurations, HEA may have similar performance to TabuCol. However, the smallest distance reached by TabuCol can be linked, at least partially, to the higher success rate of TabuCol in this region when compared to other methods, as demonstrated in Figs. 3 and 4.

In the region shown in Fig. 5 (c), where graph energy ranges from $[7, 10)$, there is no sign of search stagnation, suggesting that perhaps better solutions would be obtained if these instances were granted further execution time. At the end of the 5×10^{10} constraint checks AntCol reaches the smallest average distance using the I_{AC}=32 iteration limit. The Friedman pairwise comparison is inconclusive in the grey band, suggesting that TabuCol with T_{TC}=Reactive, some configurations of HEA parameters, and other AntCol parameters may have similar performance. It is, however, interesting to notice that the default AntCol parameter (I_{AC}=2) falls outside the band, presenting good evidence that, for this method, parameters different from the default one perform better in this region.

At the highest energy region shown in Fig. 5 (d), which comprises instances for which $\mathcal{E}(G) > 10$, there are no signs of stagnation, once again suggesting the possibility of improving solution quality via increased run times. This upper energy region clearly favours HEA, using parameter set $\{I_{HE}$=8, P_{HE}=4, X_{HE}=GPX$\}$. The grey band illustrates the region where the Friedman pairwise comparison is inconclusive. In this case, only two other configurations of HEA parameters, namely $\{I_{HE}$=8, P_{HE}=4, X_{HE}=GPX+Kmp$\}$ and $\{I_{HE}$=16, P_{HE}=4, X_{HE}=GPX$\}$, fall inside the band, none of them being the default.

Other problem instance features like σ_{CB} and $\rho(G)$ also appear to have influence on the dynamic behaviour of the algorithms. Extreme values of both σ_{CB} and $\rho(G)$ (low and high) apparently require less runtime than intermediate values. On the other hand, $\mathcal{E}(G)$ appears to have a more linear influence (i.e.: low $\mathcal{E}(G)$ requires less resources, while high $\mathcal{E}(G)$ requires more), therefore the analysis presented here is focused solely on the energy.

In summary, it seems that the necessary runtime demanded by the algorithms in order to converge or stagnate can be estimated based on instance features, particularly the energy. For extremely low energy graphs ($\mathcal{E}(G) < 2$) it seems that a runtime limit as low as 10^8 constraint checks would suffice, since little improvement is observed after that. Intermediate values of this feature ($\mathcal{E}(G) \in [2, 7)$) suggest that most algorithms would be already stagnating at 10^{10} constraint checks. However, the appropriate runtime required by higher energy instances ($\mathcal{E}(G) > 7$) cannot be determined from the data collected in the experiments described here. Further experiments, possibly requiring high volumes of computational resources would have to be conducted to clarify this point.

5.3 Making an informed decision on the best algorithm for each problem instance

The combination of the σ_{CB} and $\mathcal{E}(G)$ dimensions may reveal even more information about which specific instance pockets are more favourable to certain parameter configurations. In this section a simple methodology for selecting the best performing algorithm for specific regions of the instance space is adopted. It consists of slicing the two dimensional instance space defined by σ_{CB} and $\mathcal{E}(G)$, arbitrarily creating a 3×4 grid composed of 12 regions. For each region, shown in Fig. 6, the best performing algorithm is chosen based on the frequency of the best known solutions found. These frequencies are then shown in Fig. 6, represented as the number of problem instances for which the algorithm associated to the respective region reached $\overline{\chi}(G)$ over the total instances in that region. Occasional ties between algorithms are resolved by looking at the distance and algorithm failure metrics.

Summarising, the best overall performing method shown in Table 3, considering only default parameters, was HEA, reaching $\overline{\chi}(G)$ for 70.1% of the instances. If tuned parameter configurations are taken into account, then the best overall choice (HEA, $I_{HE} = 8$, $P_{HE} = 4$, $X_{HE} = GPX$) would have reached $\overline{\chi}(G)$ in 78.8% of the instances. On the other hand, if the algorithms were selected based on problem instance features according to Fig. 6, the best known solution would have been reached for 83.2% of the instances. That would amount to about 13% of the graphs being coloured using fewer colours, compared to the best overall method using the default parameters.

For the sake of comparison, in the work of Smith-Miles et al (2014) a powerful machine learning technique is used to predict the regions of the instance space where each heuristic method would perform best. That approach is reported to have an accuracy ranging from 73% to 90%, depending on the method under consideration, being 82% for HEA specifically. The simple slicing approach described here has pointed to an algorithm that would have reached $\overline{\chi}(G)$ for 98% of the instances in some regions ($\mathcal{E}(G) < 2$), but would have fairly poor performance (38%) in extreme regions ($\{\mathcal{E}(G) > 10\} \cap \{\sigma_{CB} < 5\}$). Since only 50 instances of those available actually fall in that region (less than 1% of the total), this may not be enough to actually point to a clear favourite algorithm.

On the other hand, the remaining higher energy region ($\{\mathcal{E}(G) > 10\} \cap \{\sigma_{CB} \geq 5\}$, containing around 1,000 instances) is dominated by the HEA method with a specific parameter set, reaching $\overline{\chi}(G)$ in more than 80% of the instances. This result seems to conflict with the findings of Smith-Miles et al (2014), which stated that extreme values of energy are more favourable to the AntCol method, outperforming HEA – at least when the default

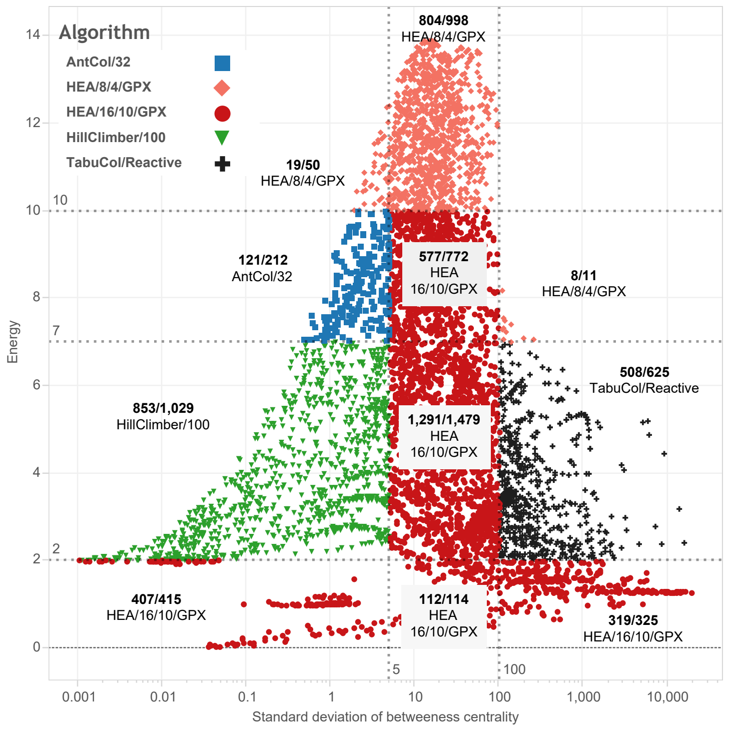

Figure 6 Distribution of the best performing algorithms based on two proposed features of the instance space.

parameters are used. The results of this experiment however suggest that a proper parameter setup can circumvent the apparent weakness of HEA in that particular region.

One important consideration is that the association of algorithms with regions of the instance space proposed here is valid for the particular computational resource budget (iteration limit) used in this experiment. As shown in Fig. 5, evidence suggests that, at least for certain regions of the instance space, the algorithms would exhibit further improvement in solution quality if allowed to run longer. In this case, it is also possible that the rank of best performers would be affected, but this issue requires further investigation.

6 Conclusions and Future Work

This paper has evaluated the effects of parameter tuning on a collection of heuristic methods for the graph colouring problem. Specific values of problem instance features have been associated with specific parameter configurations exhibiting superior performance. Three relevant features of the graphs influencing the heuristics performance under different parameter configurations have been identified: standard deviation of betweenness centrality, energy, and graph density.

Over 6,000 problem instances with diverse feature values were used to evaluate the performance of five heuristic methods and different parameter configurations, resulting in over 260,000 algorithm executions. The results strongly suggest that selecting and tuning the heuristics according to the features of problem instances leads to an average superior performance when compared to any single algorithm applied to the whole instance space. Applying simple criteria for partitioning the instance space into regions and associating one specific method and parameter values to each region has lead to a 13% increase in the number of best solutions found when compared to the default setup of HEA, which is the method regarded as the best overall performer among the set of heuristics used in this work (Lewis et al, 2012; Smith-Miles et al, 2014).

Concerning the particular region of the instance space composed of high energy graphs, a considerably superior performance is obtained by using HEA under a specific parameter configuration different from the default one. In a previous work (Smith-Miles et al, 2014) HEA has been regarded as relatively weak in this region of the instance space, at least when using the default setup. We may also conclude that HEA shows a more robust overall performance when using one particular set of parameters among those we have experimented with. The "8/4/GPX" combination appears to be superior to any other single combination of HEA parameters. Our observation highlights the powerful effect of parameter tuning in heuristic methods, which has not been thoroughly explored by those previous works.

The performance of other methods has also been strongly influenced by parameter tuning, potentially increasing manyfold the performance measure values relative to the default parameter configuration (like the case of AntCol method). A similar conclusion has been previously reported for linear solvers (Hutter et al, 2013). On the other hand, the tabu search and hill climbing-based heuristics have been shown to be more robust regarding parameter tuning, at least with respect to the features considered in this work.

Additionally, a metric of algorithm failure rate has been applied to identify regions of the instance space where the algorithms fail to produce feasible candidate solutions. The AntCol method shows surprisingly high failure rates with low density instances ($\rho(G) < 0.3$), particularly when using low iteration limits. TabuCol and PartialCol show high failure rates with low energy instances ($\mathcal{E}(G) < 2$), regardless of the tabu strategy adopted.

Another important contribution of this work is the assessment of the convergence/stagnation speed of the heuristic methods and the computational burden required to solve a given problem instance. We have presented evidence that this strongly depends on certain problem instance features, especially the energy, regardless of the heuristic method being used. We believe that an approach for estimating resource budget demanded by a given GCP instance based on its features could be established; however, detailing and validating that approach would demand additional research.

In summary, the human-interpretable knowledge extracted from the experimental results described in this paper can be promptly used by practitioners to fine tune their methods. It can also help algorithm designers to identify and address weaknesses in their designs or implementations, like the high algorithm failure rates observed for certain methods when applied to problem instances exhibiting very specific features.

Regarding future work, it would be useful to investigate: (i) The tuning of other parameters not externally accessible in the software suite used in this work, but inherent to the underlying metaheuristics like Ant Colony (e.g. α and β, etc.), which could improve even further the performance in specific regions of the instance space; (ii) The issue of high failure rate of some algorithms in specific regions of the instance space, which may point to either pitfalls in the algorithm design or to problems in the software implementation; (iii) The application of machine learning techniques to extract models that could be useful for selecting and tuning the heuristic methods applied to the GCP; (iv) An extension of the analysis to other combinatorial optimisation problems and associated heuristic methods.

Note

[1]Incumbent solution, as defined by Hoos and Stützle (2005), is the best candidate solution found at a given execution stage of an algorithm.
[2]We tried to reproduce an instance set as similar as possible to the one used in the experiments of Smith-Miles et al (2014). Some of the instances were downloaded from the web, some were generated following the descriptions provided in (Smith-Miles et al, 2014) and (Lewis et al, 2012), while others were provided to us by the corresponding authors of these two papers.
[3]The scrips used for instance generation can be downloaded from `http://www.rhydlewis.eu/resources/gColParam.zip`. They are meant to be used with the Culberson's Generator (Culberson, 2002) and Networkx (`https://networkx.github.io/`)
[4]The software suite is available at `http://www.rhydlewis.eu/resources/gCol.zip`.
[5]Both these strategies consist in on-line parameter tuning techniques already built into the original implementation.
[6]Values of ϕ and η are are alternated during the search in the range of $[500; 10,000]$ and $[5; 15]$ respectively.
[7]This is the definition used by Smith-Miles et al (2014). Other references define it as "the sum of the absolute values of the eigenvalues of the adjacency matrix"(Balakrishnan, 2004).
[8]A dataset containing the experimental data is available at `http://www.rhydlewis.eu/resources/gColParam.zip`
[9]https://www.debian.org/
[10]https://www.gnu.org/software/parallel/

References

Andersson M, Bandaru S, Ng AH (2016) Tuning of multiple parameter sets in evolutionary algorithms. In: Proceedings of the Genetic and Evolutionary Computation Conference 2016, GECCO '16, pp 533–540

Aranha C, Junior J, Kanoh H (2018) Comparative study on discrete si approaches to the graph coloring problem. In: GECCO âŁ™18: Proceedings of the Genetic and Evolutionary Computation Conference Companion, Association for Computing Machinery, New York, NY, USA, pp 81–82, DOI 10.1145/3205651.3205664

Balakrishnan R (2004) The energy of a graph. Linear Algebra and its Applications 387:287 – 295, DOI http://dx.doi.org/10.1016/j.laa.2004.02.038, URL `http://www.sciencedirect.com/science/article/pii/S0024379504001259`

Birattari M (2009) Tuning Metaheuristics: A Machine Learning Perspective, 1st edn. Springer Publishing Company, Incorporated

Blöchliger I, Zufferey N (2008) A graph coloring heuristic using partial solutions and a reactive tabu scheme. Comput Oper Res 35(3):960–975, DOI 10.1016/j.cor.2006.05.014, URL `http://dx.doi.org/10.1016/j.cor.2006.05.014`

Box GEP, Hunter JS, Hunter WG (2005) Statistics for experimenters : design, innovation, and discovery. Wiley series in probability and statistics, Wiley-Interscience

Culberson J (2002) A Graph Generator for Various Classes of k-Colorable Graphs. URL `http://webdocs.cs.ualberta.ca/~joe/Coloring/Generators/generate.html`

Demšar J (2006) Statistical comparisons of classifiers over multiple data sets. Journal of Machine Learning Research 7:1–30

Dowsland KA, Thompson JM (2008) An improved ant colony optimisation heuristic for graph colouring. Discrete Applied Mathematics 156(3):313 – 324, DOI http://dx.doi.org/10.1016/j.dam.2007.03.025, URL `http://www.sciencedirect.com/science/article/pii/S0166218X07001321`, combinatorial Optimization 2004CO2004

Eiben AE, Smit SK (2011) Evolutionary algorithm parameters and methods to tune them. In: Autonomous search, Springer, pp 15–36, DOI 10.1007/978-3-642-21434-9_2, URL `http://dx.doi.org/10.1007/978-3-642-21434-9_2`

Galinier P, Hao JK (1999) Hybrid evolutionary algorithms for graph coloring. Journal of Combinatorial Optimization 3(4):379–397, DOI 10.1023/A:1009823419804, URL `http://dx.doi.org/10.1023/A%3A1009823419804`

Hardy B, Lewis R, Thompson J (2017) Tackling the edge dynamic graph colouring problem with and without future adjacency information. Journal of Heuristics DOI 10.1007/s10732-017-9327-z, URL `https://doi.org/10.1007/s10732-017-9327-z`

Hertz A, Plumettaz M, Zufferey N (2008) Variable space search for graph coloring. Discrete Appl Math 156(13):2551–2560, DOI 10.1016/j.dam.2008.03.022, URL `http://dx.doi.org/10.1016/j.dam.2008.03.022`

Hoos H, Stützle T (2005) Stochastic Local Search: Foundations and Applications. Morgan Kaufmann Series in Artificial Intelligence, Morgan Kaufmann Publishers

Hutter F, Hoos H, Leyton-Brown K (2013) Identifying key algorithm parameters and instance features using forward selection. In: Nicosia G, Pardalos P (eds) Learning and Intelligent Optimization, Lecture Notes in Computer Science, vol 7997, Springer Berlin Heidelberg, pp 364–381

Levine DM, Berenson ML, Stephan D, Krehbiel TC, Ng PT (2007) Statistics for Managers Using Microsoft Excel (5th Edition). Prentice-Hall, Inc., Upper Saddle River, NJ, USA

Lewis R (2009) A general-purpose hill-climbing method for order independent minimum grouping problems: A case study in graph colouring and bin packing. Comput Oper Res 36(7):2295–2310, DOI 10.1016/j.cor.2008.09.004, URL `http://dx.doi.org/10.1016/j.cor.2008.09.004`

Lewis R (2015) A Guide to Graph Colouring: Algorithms and Applications. Springer International Publishing

Lewis R, Thompson J, Mumford C, Gillard J (2012) A wide-ranging computational comparison of high-performance graph colouring algorithms. Comput Oper Res 39(9):1933–1950, DOI {10.1016/j.cor.2011.08.010}

Liu H, Motoda H (1998) Feature Selection for Knowledge Discovery and Data Mining. Kluwer Academic Publishers, Norwell, MA, USA

Malaguti E, Monaci M, Toth P (2008) A metaheuristic approach for the vertex coloring problem. INFORMS J on Computing 20(2):302–316, DOI 10.1287/ijoc.1070.0245, URL `http://dx.doi.org/10.1287/ijoc.1070.0245`

Moalic L, Gondran A (2017) Variations on memetic algorithms for graph coloring problems. Journal of Heuristics DOI 10.1007/s10732-017-9354-9, URL `https://doi.org/10.1007/s10732-017-9354-9`

Pedersen TB, Jensen CS (2001) Multidimensional database technology. Computer 34(12):40–46, DOI 10.1109/2.970558, URL `http://dx.doi.org/10.1109/2.970558`

Porumbel DC, Hao JK, Kuntz P (2010) An evolutionary approach with diversity guarantee and well-informed grouping recombination for graph coloring. Comput Oper Res 37(10):1822–1832, DOI 10.1016/j.cor.2010.01.015, URL `http://dx.doi.org/10.1016/j.cor.2010.01.015`

Rice JR (1976) The Algorithm Selection Problem. Advances in Computers 15:65–118

Smith-Miles K, Baatar D, Wreford B, Lewis R (2014) Towards objective measures of algorithm performance across instance space. Comput Oper Res 45:12–24

Smith-Miles KA (2008) Towards insightful algorithm selection for optimisation using meta-learning concepts. In: 2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence), pp 4118–4124

Smith-Miles KA (2009) Cross-disciplinary perspectives on meta-learning for algorithm selection. ACM Comput Surv 41(1):6:1–6:25, DOI 10.1145/1456650.1456656, URL `http://doi.acm.org/10.1145/1456650.1456656`

Stützle T, Fernandes S (2004) New benchmark instances for the qap and the experimental analysis of algorithms. In: Gottlieb J, Raidl G (eds) Evolutionary Computation in Combinatorial Optimization, Lecture Notes in Computer Science, vol 3004, Springer Berlin Heidelberg, pp 199–209, DOI 10.1007/978-3-540-24652-7_20, URL `http://dx.doi.org/10.1007/978-3-540-24652-7_20`

Talbi EG (2009) Metaheuristics: From Design to Implementation. Wiley Publishing

Titiloye O, Crispin A (2011) Quantum annealing of the graph coloring problem. Discret Optim 8(2):376–384, DOI 10.1016/j.disopt.2010.12.001, URL `http://dx.doi.org/10.1016/j.disopt.2010.12.001`

Wolpert DH, Macready WG (1997) No free lunch theorems for optimization. Trans Evol Comp 1(1):67–82

Wu Q, Hao JK (2012) Coloring large graphs based on independent set extraction. Comput Oper Res 39(2):283–290, DOI 10.1016/j.cor.2011.04.002, URL `http://dx.doi.org/10.1016/j.cor.2011.04.002`