Applied Soft Computing 109 (2021) 107492

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

Search trajectory networks: A tool for analysing and visualising the behaviour of metaheuristics

Gabriela Ochoa [a,*], Katherine M. Malan [b], Christian Blum [c]

[a] Computing Science and Mathematics, University of Stirling, Stirling, Scotland, UK
[b] Department of Decision Sciences, University of South Africa, South Africa
[c] Artificial Intelligence Research Institute (IIIA-CSIC), Bellaterra, Spain

ARTICLE INFO

Article history:
Received 20 July 2020
Received in revised form 12 March 2021
Accepted 5 May 2021
Available online 14 May 2021

Keywords:
Algorithm analysis
Search trajectories
Complex networks
Continuous optimisation
Combinatorial optimisation
Visualisation

ABSTRACT

A large number of metaheuristics inspired by natural and social phenomena have been proposed in the last few decades, each trying to be more powerful and innovative than others. However, there is a lack of accessible tools to analyse, contrast and visualise the behaviour of metaheuristics when solving optimisation problems. When the metaphors are stripped away, are these algorithms different in their behaviour? To help to answer this question, we propose a data-driven, graph-based model, *search trajectory networks* (STNs) in order to analyse, visualise and directly contrast the behaviour of different types of metaheuristics. One strength of our approach is that it does not require any additional sampling or algorithmic methods. Instead, the models are constructed from data gathered while the metaheuristics are solving the optimisation problems. We present our methodology, and consider in detail two case studies covering both continuous and combinatorial optimisation. In terms of metaheuristics, our case studies cover the main current paradigms: evolutionary, swarm, and stochastic local search approaches.

1. Introduction

The last few decades have seen the introduction of a large number of "novel" metaheuristics inspired by different natural and social phenomena. Sörensen [1] argues that this development has taken the field a step backwards, rather than forwards, and that instead of more new methods, we need critical evaluation of established methods to reveal their underlying mechanics. There have been attempts to describe algorithms using standard metaphor-free terminology [2,3]. Although this helps in understanding the mechanisms of algorithms and in highlighting similarities and differences, it still does not provide insight into the resulting search behaviour.

The behaviour of metaheuristics is often described in relation to the level of exploration or exploitation, or the broader concept of intensification/diversification (I&D) [4]. However, there is no generally accepted understanding of this concept in the evolutionary computing research community [5] and no general way of analysing or measuring the level of I&D of metaheuristics. Convergence analysis (measured through diversity of solutions) of population-based algorithms is related to I&D and is one way of describing algorithm behaviour. However, knowing when a population converges ignores *where* in the search space this is happening and hence whether convergence is premature or not. Convergence analysis is also not applicable to single-point metaheuristics.

Others have proposed techniques for visualising the behaviour of search algorithms [6–9]. These approaches use dimensionality reduction to map search spaces to two or three dimensions and in this way track search progress. Our proposed STN model is similar in aim to these approaches, except that STNs are graph objects with nodes and edges that can be analysed and visualised, rather than the full search space reduced to a visualisable Cartesian plane.

Many natural and technological systems are composed of a large number of highly interconnected units; examples are neural networks, biological and chemical systems, social interacting species, the Internet and the World Wide Web. A key approach to capture the global properties of such systems is to model them as graphs whose nodes represent the units, and whose links stand for the interactions between them. This simple, yet powerful concept has been used to study a variety of complex systems where the goal is to analyse the pattern of connections between components in order to understand the behaviour of the system. Once a system is modelled as a network, an extensive set of mathematical and computational tools is available for analysing, understanding and visualising the system [10,11]. We argue that understanding and contrasting the behaviour of metaheuristics is a complex task, and thus complex systems tools are paramount.

* Corresponding author.
E-mail address: gabriela.ochoa@stir.ac.uk (G. Ochoa).

https://doi.org/10.1016/j.asoc.2021.107492
1568-4946/

We therefore propose a data-driven network model to analyse and visualise the behaviour of metaheuristics.

We initially presented the concept of *search trajectory networks* in a recent conference paper [12], where we modelled the dynamics of two population-based algorithms when solving continuous benchmark functions. Here, we extend and generalise the concepts, methodology and computational experiments to cover not only population-based approaches, but also stochastic local search methods (also called single-point metaheuristics) and both continuous and combinatorial optimisation. We emphasise the use of this modelling technique to directly contrast the behaviour of different types of metaheuristics in a unified model that can be analysed quantitatively and visually. Since constructing, analysing and visualising the networks are key contributions of this article, our STNs repository[1] provides the required source code (R scripts) and example datasets to create, merge, analyse and visualise the STN models for both continuous and discrete optimisation problems.

The outline of this paper is as follows. Section 2 gives an overview of related work. Section 3 gives the relevant definitions behind search trajectory networks (STNs), describes the approach to construct the data-driven models and presents a simple illustrative example. Thereafter, two case studies are thoroughly presented to illustrate the application of STNs to contrast and understand the behaviour of different types of metaheuristics when solving continuous (Section 4) and combinatorial (Section 5) optimisation problems, respectively. Finally, our concluding remarks and suggestions for future work are discussed in Section 6.

2. Related work

The initial inspiration for STNs came from the study of local optima networks (LONs) [13,14], which are a compressed model of fitness landscapes where nodes are local optima and edges represent possible transitions between optima with a given search operator. LONs in turn were inspired by network-based models of energy landscapes in computational chemistry [15]. *Disconnectivity graphs* [16,17], also known as *barrier trees* [18], are another graph-based modelling tool originated from the study of energy landscapes, which has also been applied to model the fitness landscapes of optimisation problems [19]. STNs differ from these tools as the goal is not to model the structure of fitness landscapes, but instead the search behaviour of optimisation algorithms.

A recent body of work has used networks to understand the dynamics of population based algorithms. The idea, as initially proposed by Zelinka and Davendra [20], is to use graphs whose connections represent interactions amongst the individuals during all generations; vertices are individuals that are activated by other individuals, incrementally from generation to generation. Follow-up work has studied differential evolution [21,22] and particle swarm optimisation methods using this approach [23,24], where the emphasis is to model the communication or influence of individuals (or particles) inside the population or swarm. These network models have been shown to be useful for visualising the behaviour and capturing the trade-off between exploration and exploitation of the studied algorithms. STNs differ from these approaches as the main goal is not to model the interactions among candidate solutions in the population, but instead to model the trajectories of representative solutions across the search process. Moreover, STNs can be applied to any metaheuristic, not only to population-based ones, and merged STNs allow us to directly contrast the behaviour of different metaheuristics.

[1] https://github.com/gabro8a/STNs.git

In population-based algorithms, the way in which diversity changes over time can be seen as an approach to characterise algorithm behaviour. Bosman and Engelbrecht [25] proposed a single numerical measure called *diversity rate of change* for characterising the exploration–exploitation trade-off in particle swarms. Their premise was that the profile of the reduction in diversity (measured using the average Euclidean distance around the centre of the swarm [26]) could be captured by the slopes of a two-piecewise linear approximation of the diversity over time. Although diversity provides one important view of algorithm behaviour, it ignores where in the search space the population is moving and hence whether convergence is premature or not.

In the domain of multi-objective optimisation involving more than three objectives, there is a body of literature involving the visualisation of Pareto front approximations [27]. These approaches involve dimensionality reduction techniques to show algorithm progression in improving objective values over time. Trace generation plots [28] are similar in that they show how the hypervolume value changes over generations. Although these visualisation approaches can help in understanding and contrasting algorithm behaviour, they differ from STNs as they are visualising objective space, rather than solution space.

A different approach to tracking search dynamics is to map multi-dimensional solutions into lower dimensions to visualise how trajectories of solutions change over time. Dimensionality reduction techniques that have been used for this aim include principal component analysis [6], Sammon mapping [7], and t-distributed stochastic neighbour embedding [8]. With this approach, the positions of solutions relative to each other and the movement of individuals in a population can be visualised over an algorithm run using a sequence of 2-D frames or a 3-D stacking of 2-D frames [9]. STNs are similar to these approaches in that they also provide a visualisation of search dynamics and trajectories, but the main difference is that the location information and movement through the search space is captured in a graph object, allowing the information to be analysed using a wealth of mathematical and visualisation tools. Our approach also includes the ability to analyse the information at different levels of granularity of the search space by simply changing the definition of a location in the model.

3. Search Trajectory Networks (STNs)

3.1. Definitions

In order to define a network model, we need to specify the nodes and edges. The relevant definitions are given below.

Representative solution. A solution to the optimisation problem at a given time step that represents the status of the search algorithm based on predefined criteria. For example, in a population-based algorithm, the best solution in the population at the given iteration might be chosen as the representative solution, whereas in a single-point method the incumbent solution is the obvious choice for a representative solution.

Location. A non-empty subset of solutions that results from a predefined partitioning of the search space. Each solution in the search space is an element of one and only one location. Each location is assigned a representative objective value using predefined criteria. In discrete search spaces, a location can be modelled as a single solution.

Search trajectory. Given a sequence of representative solutions in the order in which they are encountered during the search process, a search trajectory is defined as a sequence of locations formed by replacing each solution with its corresponding location. The frequency of recording the representative solutions in the trajectory is specified using predefined criteria.

Node. A location in a search trajectory of the search process being modelled. The set of nodes is denoted by N.

Edges. Edges are directed and connect two consecutive locations in the search trajectory. Edges are weighted with the number of times a transition between two given nodes occurred during the process of sampling and constructing the STN. The set of edges is denoted by E.

Search Trajectory Network (STN). An STN is a directed graph $\text{STN} = G(N, E)$, with node set N, and edge set E as defined above.

3.2. Sampling and model construction

One strength of our approach is that it does not require implementing any specific sampling or data gathering method to construct the models. Instead, the data to construct the models is gathered while the algorithm under study is running. Specifically, the required output from a run of the algorithm is a list of steps (edges) connecting two adjacent representative solutions in the search process. Each search step (algorithm iteration) is stored as an entry in a log file containing the two consecutive representative solutions being linked with the step. Both the encoding (solution vector) and evaluation (fitness value) of each representative solution in a step are stored.

STN model. Once the data logs of a predefined number of runs of a given algorithm-problem instance pair are gathered, a post-processing step aggregates all the representative solutions and transitions to construct a single network object. A mapping between the representative solutions and their unique locations and objective values is required. For minimisation problems, the representative objective value is the minimum objective value of all solutions that visited a location across all runs. The mapping from solutions to locations depends on the search space under consideration. Examples of such mappings in continuous and discrete search spaces are given in the case studies (Sections, Sections 4 and 5), respectively. When constructing the network models, counters are kept as attributes for both nodes and edges to account the number of times they were visited during the sampling process.

Merged STN model. Once the STN models for a set of algorithm–instance pairs are constructed, we can proceed to merge the STNs of different algorithms for a given problem instance. Let us assume we have two algorithms. The merged STN model of the two algorithms for a given instance is obtained by the graph union of the two individual graphs for that instance. More formally, let $\text{STN}_\text{A} = G(N_\text{A}, E_\text{A})$ and $\text{STN}_\text{B} = G(N_\text{B}, E_\text{B})$ be the STNs of algorithms A and B for a given instance. We then construct STN_merged as the union of the two graphs. Specifically, $\text{STN}_\text{merged} = G(N_\text{A} \cup N_\text{B}, E_\text{A} \cup E_\text{B})$. The merged graph contains the nodes and edges that are present in at least one of the algorithm graphs. Attributes are kept for the nodes and edges indicating whether they were visited by both algorithms or by one of them only.

3.3. Network metrics

Once a system is modelled as a graph, many structural properties can be computed. The most basic metrics are the number of nodes and edges, but a variety of other metrics could be calculated such as the degree distribution, length of paths, community structure, and centrality of nodes to name a few [10]. To keep things simple we propose five straightforward network metrics to assess the global structure of the trajectories, and thus bring insight into the difficulty of the instances and the behaviour of the metaheuristics modelled. These metrics are summarised in Table 1. It is worth noting that additional metrics could also be considered.

The justification of this selection of metrics is as follows. The total number of nodes, *ntotal*, gives an idea of the amount of the search space that was explored. The number of nodes with best-found evaluation, *nbest*, indicates whether different locations of the search space evaluate to best-found fitness. The number of nodes at the end of trajectories (different than the best nodes), *nend*, indicates how likely it is for trajectories to end up in sub-optimal locations. The number of shared nodes, *nshared*, indicates whether there are solutions or areas of the search space that tend to attract the trajectories of different algorithms.

The degree of a node in a graph is simply the number of edges connected to it. In directed graphs, such as STNs, we can distinguish incoming and outgoing edges, and thus incoming and outgoing degrees. Moreover, when edges are weighted such as in STNs, it is customary to use the weighted degree of a node, also called *strength* in graph theory terminology, which is based on the number of edges connected to the node, but ponderated by the weight of each edge.

Our final metric (*best-strength*) computes the incoming weighted degree of the best node(s). When there is more than one best node, this metric simply sums their incoming strengths. In order to have values between zero and one, we normalise this metric by the number of algorithm runs used to sample and construct the STN model(s). This metric evaluates to one when all runs end in a best found solution. Note that *best-strength* provides a measure of the centrality and reachability of the best-found solution(s). It is worth noting that the centrality of good solutions has been found to correlate with search difficulty in the study of local optima networks [29,30].

3.4. Network visualisation

Visualisation is a powerful tool that may allow us to appreciate structural features which can be difficult to infer from the network metrics alone. The most common visual representation of a network, which we have used throughout this paper, is the so-called node–edge diagram. Node–edge diagrams assign the nodes to points in the two-dimensional or three-dimensional Euclidean space, and connect adjacent nodes by straight lines or curves. Arrowheads are used to indicate the direction of connections in the context of directed graphs. Nodes are then drawn on top of the edges using simple geometric shapes. Moreover, the most important attributes of nodes and edges are assigned to visual properties (such as size and colour) of the shapes and lines; for instance, the area of a circle can be made proportional to the degree of the node in order to highlight hubs (i.e. highly connected nodes).

The graph visualisations in this paper were produced with the igraph library [31] of the R programming language. We considered *force-directed* layout algorithms, such as Fruchterman–Reignold [32] and Kamada–Kawai [33]. Force-directed layout algorithms are based on physical analogies and do not rely on any assumptions about the structure of the networks. These algorithms strive to satisfy the following generally accepted criteria [32]:

- Vertices are distributed roughly evenly on the plane (a circle in the igraph implementation).
- The number of crossing edges is minimised.
- The lengths of edges are approximately uniform.
- The inherent symmetries in the networks are respected, i.e., sub-networks with similar inherent structure are usually laid out in a similar manner.

Table 1
Description of network metrics.

ntotal	Total number of nodes.
nbest	Number of nodes with the best-found evaluation.
nend	Number of nodes at the end of trajectories (other than the nodes with the best evaluation).
nshared	Number of nodes visited by more than one algorithm.
best-strength	Normalised incoming strength (weighted degree) of the best node(s).

Note that the R scripts for creating, visualising and analysing the STN models presented in this paper are provided at https://github.com/gabro8a/STNs.git. The repository also contains example datasets and a README file explaining the input format and how to use the scripts provided.

3.5. Illustrative example

To illustrate the concept of an STN and how it is constructed, we provide a simple example in continuous optimisation where the search space can be visualised alongside the STN. Fig. 1(a) shows the Schwefel 2.26 benchmark function, which is to be minimised, in two dimensions. The fitness landscape is multi-modal with a multi-funnelled global structure [34]. The global optimum is found at the upper right corner of the plot, approximately at position (420, 420).

Fig. 1(b) shows the contour plot of the problem where areas of higher fitness are shaded darker. The global optimum position is shown in the contour plot as a red dot. The example considers iterated local search (ILS) [35] – which is a simple, yet powerful, search strategy combining a perturbation stage with a hill-climbing (local search) process – and a version of differential evolution (DE) [36]. Three runs of each algorithm were executed on the problem and the trajectories are shown on the contour plot in blue for ILS and orange for DE. Initial random positions are shown as yellow filled-in squares and sub-optimal end points are shown as black filled-in triangles. Each arrow indicates the start and end of an improving iteration of ILS (perturbation from previous position followed by local search) or a change in the best individual of the DE population. The plot of the trajectories in the actual search space is quite messy and difficult to interpret. It is not that clear to see, but one of the three blue ILS runs successfully finds the global optimum and two of the orange DE runs also reach the global optimum. The two unsuccessful runs of the ILS end in the same local optimum, approximately at position $(-300, 420)$, while the one unsuccessful DE run ends in the local optimum in the bottom right corner.

Fig. 1(c) shows the merged STN constructed from the trajectories visualised in Fig. 1(b). Nodes correspond to locations and edges to transitions between them. As indicated in the legend, the colour and shape of nodes reflect their type. Yellow squares indicate the start of trajectories, while dark grey triangles the end of trajectories. The red circle distinguishes the best-found location (which in this case contains the global optimum). The remaining intermediate nodes are circles coloured by the algorithm that traversed them (orange for DE and blue for ILS). If a node is traversed by more than one algorithm, it is painted in light grey. Notice that in this small example there are no light grey nodes as the only shared node is an end node (triangle in the middle of the figure, just below the global optimum), and the decoration of end nodes has precedence over that of shared nodes. The size of nodes is proportional to their incoming strength (weighted degree). The colour of an edge indicates the algorithm that traversed it. Edges traversed by more than one algorithm are painted in grey. However, this does not happen in this small example. The width of an edge is proportional to the number of times it was traversed.

It can be seen in Fig. 1(c) that one of the ILS trajectories and two of the DE trajectories end in the best location (red node).

Table 2
STN metrics for the illustrative example representing the search process of ILS and of DE on the Schwefel 2.26 benchmark function in two dimensions. A description of the metrics can be found in Table 1.

STN model	*ntotal*	*nbest*	*nend*	*nshared*	*best-strength*
Merged	33	1	2	2	0.5
DE	23	1	1	*NA*	0.67
ILS	12	1	1	*NA*	0.33

Table 3
Scalable benchmark functions (D is the dimension)

Function	Definition and domain
Michalewicz	$f(\mathbf{x}) = -\sum_{i=1}^{D} \sin(x_i)\left(\sin(ix_i^2/\pi)\right)^{2p}, \quad x_i \in [0, \pi]$
Quadric	$f(\mathbf{x}) = \sum_{i=1}^{D} \left(\sum_{j=1}^{i} x_j\right)^2, \quad x_i \in [-100, 100]$
Rana	$f(\mathbf{x}) = \sum_{i=1}^{D} x_i \sin(\alpha)\cos(\beta) + \left(x_{(i+1) \bmod D} + 1\right)\cos(\alpha)\sin(\beta)$, $D \geq 2, \quad \alpha = \sqrt{\lvert x_{i+1} + 1 - x_i \rvert}, \quad \beta = \sqrt{\lvert x_i + x_{i+1} + 1 \rvert}$, $x_i \in [-512, 512]$
Salomon	$f(\mathbf{x}) = -\cos\left(2\pi\sqrt{\sum_{i=1}^{D} x_i^2}\right) + 0.1\sqrt{\sum_{i=1}^{D} x_i^2} + 1$, $x_i \in [-100, 100]$
Schwefel 2.26	$f(\mathbf{x}) = -\sum_{i=1}^{D} \left(x_i \sin(\sqrt{\lvert x_i \rvert})\right), \quad x_i \in [-500, 500]$

The two other ILS trajectories end in the same node (dark grey triangle at the bottom of the figure), while the third DE trajectory ends in a different location (dark grey triangle in the middle of the figure). It is interesting to see that the unsuccessful DE run ends in a location that is visited by the successful ILS trajectory before escaping from it to reach the best node. This STN visualisation is certainly a much clearer illustration of the search behaviour when compared to 1(b).

Table 2 shows the metrics described in Table 1 for the merged STN visualised in Fig. 1(c). The metrics of each individual algorithm's STN model are also shown. The *ntotal* metric indicates that DE has longer trajectories than ILS, which may be a sign of a wider exploration of the search space. The lower value of *nend* and the higher value of *best-strength* of the DE STN in comparison to the ILS STN, respectively, indicate that DE is less likely to end up in a sub-optimal solution and is more likely to reach the best solution for this instance.

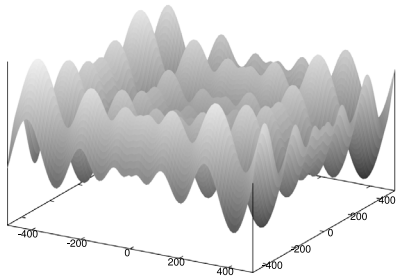
The simple example in Fig. 1 shows how an STN is an abstract visual representation of search trajectories. Note that in the context of discrete optimisation (or continuous optimisation in more than two dimensions) it is not possible to visualise the trajectories in the actual search space as was done in Fig. 1(b). STNs, however, can be used to visualise the essential information of search trajectories for any problem size to gain insight into the search behaviour of algorithms.

4. Continuous optimisation case study

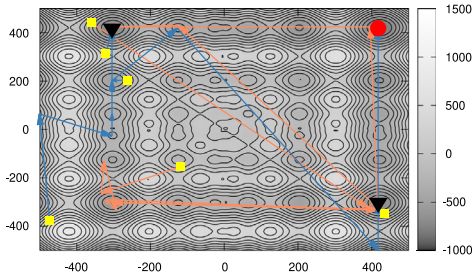
4.1. Problem formulation

The continuous optimisation problems considered are single-objective, static, bound-constrained, multivariate minimisation problems. In general such a problem can be defined as:

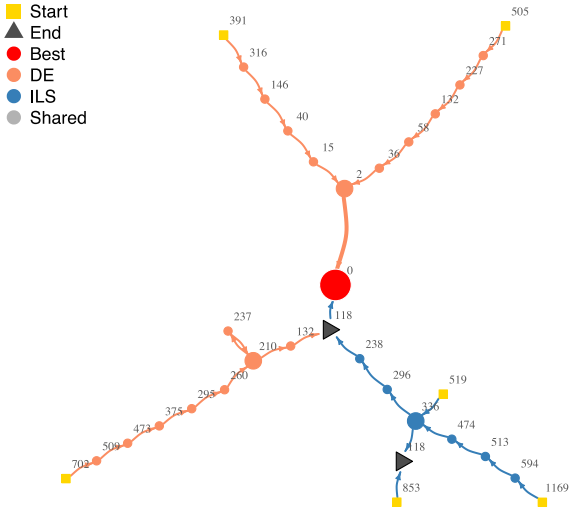
$$\min f(\mathbf{x}),\ f : \mathbb{R}^n \rightarrow \mathbb{R},\ \mathbf{x} \in \mathcal{S} \subseteq \mathbb{R}^n,$$



(a) Three dimensional view of the problem landscape. The global optimum basin can be seen in the upper right corner of the search space.



(b) Contour plot with three trajectories of ILS (blue arrows) and three of DE (orange arrows). Yellow squares denote the starting positions of the six runs, black triangles denote the sub-optimal end points and the red dot denotes the global optimum. Three of the runs (two DE and one ILS) found the global optimum while the other three were trapped in local optima.



(c) Merged STN obtained from three runs of ILS and DE. The size of nodes is proportional to their incoming degree. Node labels indicate the difference in evaluation from the optimum. Yellow squares denote the starting locations of the six runs. The red node is the best-found solution, which is reached by two DE runs and one ILS run. The two remaining ILS runs end up in the dark grey triangular node at the bottom of the figure, while the unsuccessful DE run gets trapped in the dark triangular node at the middle of the figure. The outgoing blue edge from this triangular node indicates that the successful ILS run manages to escape from it to the red node.

Fig. 1. Illustrative example of a merged STN representing the search process of ILS and of DE on the Schwefel 2.26 benchmark function in two dimensions.

where $\mathbf{x}$ is an n-dimensional candidate solution vector and $\mathcal{S}$ defines the feasible sub-region of $\mathbb{R}^n$ as defined by the domains of the variables within $\mathbf{x}$. In this study, it is assumed that $\mathcal{S}$ is defined by simple boundary constraints, which are the same for all components of the solution vector; that is,

$$x^{min} \leq x_i \leq x^{max} \quad \forall \mathbf{x} \in \mathcal{S}, \ 1 \leq i \leq n.$$

4.2. Benchmark instances

A sample of five minimisation benchmark functions (defined in Table 3) with different characteristics were chosen for demonstrating the proposed STN model in continuous spaces.

Quadric (also known as Schwefel 1.2) [37] is the only unimodal problem. Michalewicz [38] is multimodal, but also has large plateaus at high fitness values. Schwefel 2.26 [37] is multimodal and also multi-funnelled. Both Salomon [36] and Rana [36] are extremely rugged, but Salomon has a single-funnel global structure, whereas Rana has a multi-funnel structure. For the experimentation we used Rana and Salomon in 3 dimensions, Michalewicz and Schwefel 2.26 in 5 dimensions and Quadric in 10 dimensions.

4.3. Metaheuristic algorithms

For demonstration purposes, we implemented three algorithms for solving problems in continuous spaces: a swarm-based algorithm, an evolutionary algorithm, and an iterated local search (ILS) algorithm.

Particle swarm optimisation (PSO) was used for the swarm-based algorithm and differential evolution (DE) for the evolutionary algorithm. The version of PSO used in the study was traditional global best PSO [39,40] with an inertia weight term [41], 50 particles, 1.496 for both the cognitive and social acceleration

constants, and 0.7298 for the inertia weight (although the optimal choice of parameters is problem dependent, this is a common choice that works reasonably well for many problems [42]). The particular version of DE used in the study was DE/rand/1 [43], with uniform crossover, a population size of 50, a scale factor of 0.5, and a crossover rate of 0.5.

In the continuous domain, and particularly within the computational chemistry community, ILS is known as the *basin-hopping* algorithm [44]. Our implementation starts at a random location and executes a local minimisation step using a run of the limited-memory Broyden–Fletcher–Goldfarb–Shanno algorithm (L-BFGS) [45] — a quasi-Newton method that approximates the BFGS algorithm using limited memory. We used the L-BFGS implementation from the SciPy Python package [46]. From the resulting local optimum, a random perturbation is performed and the process is repeated until the budget of objective function evaluations has been reached.

We would like to clarify at this point that these three algorithms (and their parameter settings) were chosen for demonstration purposes. It is not our intention to make claims about the relative quality of the algorithms. Therefore, parameter tuning is not necessary. We are rather showing that if one search process is more successful than another one on a particular problem, the STN model can shed light on why this is the case.

4.4. Search space partitioning

As defined in Section 3, a location is a non-empty subset of solutions that results from a predefined partitioning of the search space. In this study, a continuous search space was partitioned into discrete hypercubes of a set length, defining a location as a hypercube of solutions.

A partition factor parameter (PF) was used to portion the space into hypercubes with length 10^{PF}. For example, if $PF = -2$, then the solution space was divided into hypercubes of length 10^{-2} and each location (node in the STN) was equivalent to one of these hypercubes. Solutions were mapped to locations by rounding off all components of the position to the nearest 10^{PF} to determine the identity of the enclosing hypercube.

To extract meaningful insights from the STN model, the size of partitions should be larger for larger search spaces. In this study, (assuming that the domain of values is the same for each dimension of the problem) the value for PF was expressed as a function of the range of the domain ($x_{max} - x_{min}$) and dimension (D) of the problem as follows: PF is set to $n - 2$, where n is the largest integer for which the following is true:

$$(x_{max} - x_{min}) \times D \geq 10^{n}. \quad (1)$$

For example, given a problem in three dimensions with domain $[-1, 1]$ in all dimensions, PF would be set to -2, since $2 \times 3 \geq 10^{0}$. For this problem, a location/node in the STN would be equivalent to a unique $0.01 \times 0.01 \times 0.01$ cube in the search space.

4.5. Experimental results

We implemented the three algorithms described in Section 4.3. Ten independent runs were executed of each algorithm on the five benchmark problems. Each run had a budget of $5000 \times D$ function evaluations. Representative solutions were stored for each run to form the basis of the STN graphs. For the population-based algorithms, the representative solutions were the best solutions of every iteration (the best in the population), whereas for the ILS, the representative solutions were the local optima found at the end of each run of the local search. The objective value of solutions was also stored (as a difference in value from the global optimum), rounded off to a precision of 10^{-8}. When generating the STN data, each solution was mapped to a location using the search space partitioning approach described in Section 4.4, resulting in PF values with associated partitions as shown in Table 4.

Results and discussion. For discussion purposes, Table 5 gives the performance of the three algorithms on the five problem instances. For example, we see that the ILS algorithm performed poorly on the Michalewicz problem (none of the 10 runs were able to locate the global optimum), but performed very well on Salomon (9 out of 10 runs located the global optimum). It can also be seen that although DE achieved a 0% success rate on Salomon, the average fitness difference from the origin was very low (0.06) indicating that the runs came close to the optimal fitness value.

Fig. 2 shows the merged STNs for the Rana function (3D) at three different levels of partitioning. On this instance, PSO, DE and ILS achieved success rates of 0%, 80% and 40% respectively (Table 5). The STN with medium partitioning in Fig. 2(a) shows that all except two of the DE runs (in orange) converged on the location of the best solution (which in this case contains the global optimum), corresponding with the reported success rate of 80%. Likewise, four ILS trajectories converge on the global optimum, corresponding with the 40% success rate. The three green PSO runs that appear in the cluster around the optimal value, however, can be seen to share locations with successful DE trajectories, but then end in sub-optimal locations (black triangles). The remaining PSO runs mostly explored different parts of the search space, shown as mostly unconnected green trajectories.

Insights are less clear with too coarse and too fine partitioning. In the case of the coarse partitioning in Fig. 2(b), locations are defined as hypercubes with length 100 (resulting in a partitioning of the space into $12^3 = 1728$ locations). With the larger partitions, a number of nodes in the graph merged and this results in more overlap between trajectories. It is less clear at a glance to see difference between the behaviours of the three algorithms. In the case of the fine partitioning in Fig. 2(c), locations are defined as hypercubes with length 0.1 (a partitioning of the space into 10240^3 locations). The fine partitioning results in a disjointed STN, losing some of the information on trajectories overlapping in the search space.

Table 6 provides the metrics for the merged STNs for all problem instances and Fig. 3 shows the STN metrics per algorithm as bar graphs. The metrics of merged STNs give an indication of the features and relative difficulty of the problem instances. For example, the maximal *best-strength* of Quadric with relatively low *nshared* indicates that many different paths led to the best solution. In contrast, the low *best-strength* of Rana with associated high *nend* shows the presence of many local attractors in the search space. The number of nodes in the individual algorithm STNs (seen on the left in Fig. 3) gives an indication of the lengths of the trajectories. For example, although all algorithms successfully solved Quadric, it can be seen that ILS reached the best solution in far fewer steps than PSO and DE, reflected in the lower number of nodes in the STN.

Considering the *best-strength* with the *nend* metric can also shed light on the nature of the problem. For example, although Michalewicz and Salomon have similar values for *best-strength*, the value of *nend* is significantly lower for Salomon. This means that many trajectories are ending in the same locations, indicating the presence of fewer, but stronger local-optima attractors than in the case of Michalewicz.

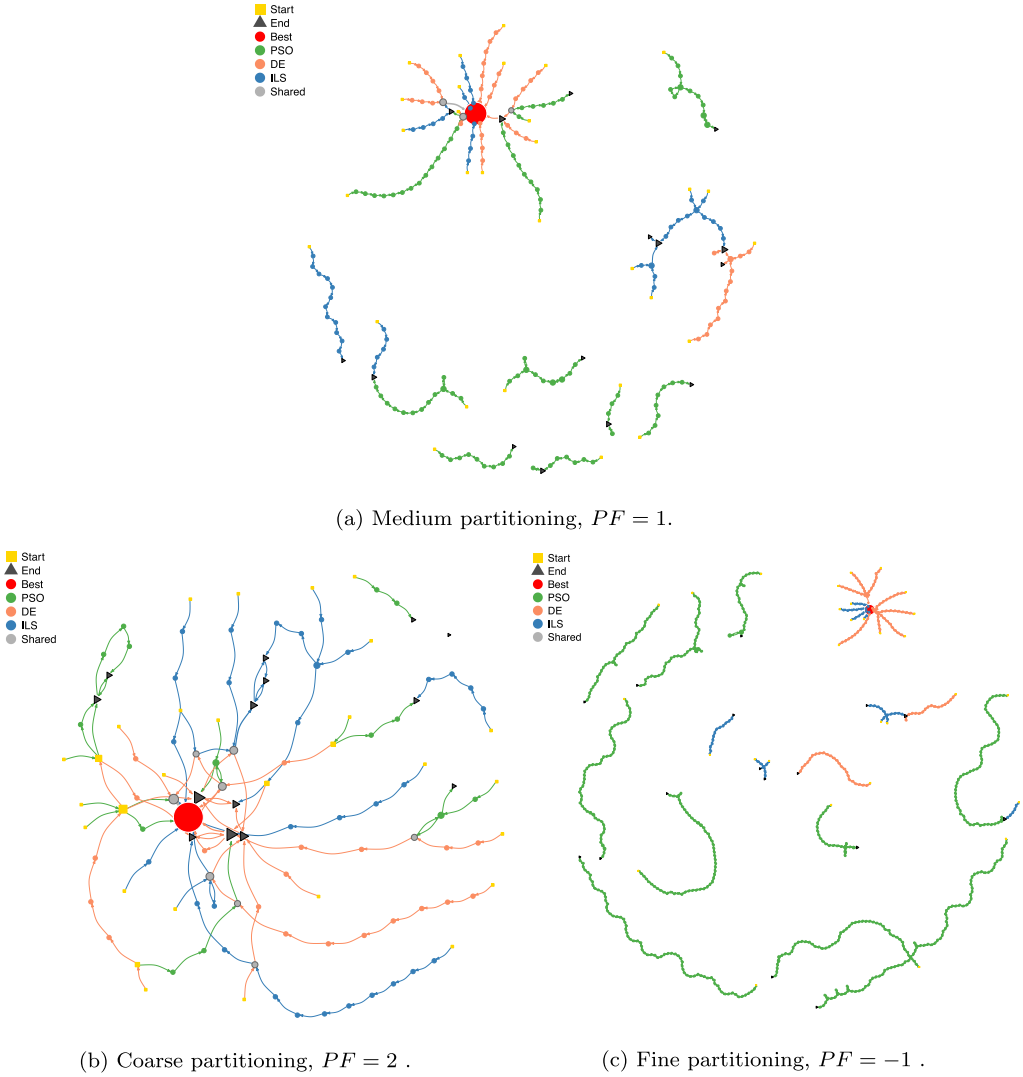
In most cases, *best-strength* corresponds with the success rate of the algorithm, but not always. For example, in the case of Salomon, the success rates of DE and ILS were 0% and 90%, respectively (Table 5). However, in Fig. 3, it can be seen that the *best-strength* of DE and ILS for Salomon are equal at 0.9. This

Table 4
Numeric benchmark instances with associated *PF* parameter values and partitions.

Instance	Dimension	Range	*PF* value	Partition
Michalewicz	5D	$x_i \in [0, \pi]$	$PF = -1$	hypercubes of length 0.1
Quadric	10D	$x_i \in [-100, 100]$	$PF = 1$	hypercubes of length 10
Rana	3D	$x_i \in [-512, 512]$	$PF = 1$	hypercubes of length 10
Salomon	3D	$x_i \in [-100, 100]$	$PF = 0$	hypercubes of length 1
Schwefel 2.26	5D	$x_i \in [-500, 500]$	$PF = 1$	hypercubes of length 10

Table 5
Performance metrics for PSO, DE, and ILS: $\overline{FD}$ (mean fitness difference from the optimum) with standard deviation ($\pm\sigma$) and SRate (percentage of runs finding the global optimum to within 10^{-4}).

Instance	PSO			DE			ILS		
	$\overline{FD}$	($\pm\sigma$)	SRate	$\overline{FD}$	($\pm\sigma$)	SRate	$\overline{FD}$	($\pm\sigma$)	SRate
Michalewicz	0.04	(±0.05)	40%	0.00	(±0.01)	90%	2.69	(±0.78)	0%
Quadric	0.00	(±0.00)	100%	0.00	(±0.00)	100%	0.00	(±0.00)	100%
Rana	150.28	(±95.75)	0%	7.65	(±16.90)	80%	83.42	(±85.36)	40%
Salomon	0.08	(±0.04)	20%	0.06	(±0.03)	0%	0.00	(±0.00)	90%
Schwefel 2.26	165.81	(±114.42)	20%	0.00	(±0.00)	100%	355.32	(±124.84)	0%



(a) Medium partitioning, $PF = 1$.

(b) Coarse partitioning, $PF = 2$.

(c) Fine partitioning, $PF = -1$.

Fig. 2. Merged STNs for the Rana benchmark, showing three search space partitioning levels.

means that although 90% of the runs for both algorithms are reaching the partition containing the global optimum, the runs of DE do not quite reach the global optimum (to within 10^{-4} as defined for the success rate measure). This is also reflected in the low $\overline{FD}$ value for DE on Salomon in Table 5.

In contrast to Rana, the STN of Salomon in Fig. 4(a) provides a different pattern of search behaviour. Although the landscape of Salomon is very rugged, it has a single funnel global structure [36]. The STN shows that all algorithms are drawn in the same direction towards the region of the global optimum. To

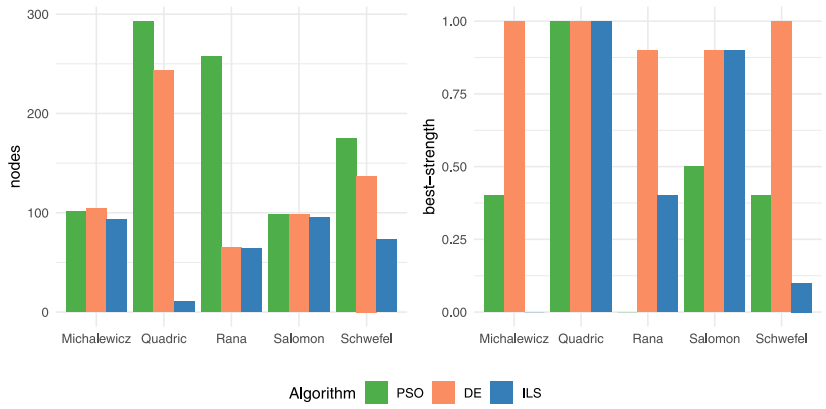


Fig. 3. Algorithm-specific metrics for the continuous problem instances. A description of the metrics is found in Table 1.

Table 6
Merged STN metrics for the continuous problem instances (using partition factors as given in Table 4). A description of all metrics is found in Table 1.

	ntotal	*nbest*	*nend*	*nshared*	*best-strength*
Michalewicz	288	1	14	10	0.67
Quadric	539	1	0	7	1.00
Rana	354	1	15	10	0.43
Salomon	260	1	4	22	0.77
Schwefel 2.26	376	1	10	8	0.57

better see the detail around the global optimum, Fig. 4(b) provides a zoomed visualisation of the STN, plotting the sub-graph containing the set of the best 25% of the nodes. It can be seen that there are many shared nodes between the different algorithms (light grey nodes) and that although some runs end in the best location, a number of runs end in the same sub-optimal end points (black triangles). The landscape of Salomon has been described as resembling "a pond with ripples" [36] around the global optimum and Fig. 4(b) is showing the trajectories converging on the "ripples" around the optimum.

5. Combinatorial optimisation case study

5.1. Problem formulation

As a showcase for combinatorial optimisation we chose the well-known p-median problem, a classic facility location problem [47]. In the p-median problem, the goal is to locate p facilities among $n > p$ demand points. After locating the facilities, each demand point is allocated to the closest (or cheapest) facility. Hereby, $d_{ij} \geq 0$ is the distance (or travel cost) between demand points $i, j \in \{1, \ldots, n\}$. The minimisation objective is to locate the p facilities such that the sum of the distances (travel costs) is minimised. The p-median problem can be expressed in terms of an integer linear programme (ILP) in the following way.

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} x_{ij}$$

$$\text{s.t.} \sum_{j=1}^{n} x_{ij} = 1 \qquad i = 1, \ldots, n$$

$$x_{ij} \leq y_j \qquad i, j = 1, \ldots, n$$

$$\sum_{j=1}^{n} y_j = p$$

$$x_{ij} \in \{0, 1\} \qquad i, j = 1, \ldots, n$$

$$y_j \in \{0, 1\} \qquad j = 1, \ldots, n$$

Algorithm 1 Solution construction (p-median problem)

```
S := ∅
while |S| < p do
    j* := Choose(S)
    S := S ∪ {j*}
end while
output: S
```

Hereby, y_j is a binary variable that indicates if a facility is located in demand point j, $j = 1, \ldots, n$. Moreover, x_{ij} is a binary variable that indicates if, or not, demand point i is allocated to facility j.

5.2. Benchmark instances

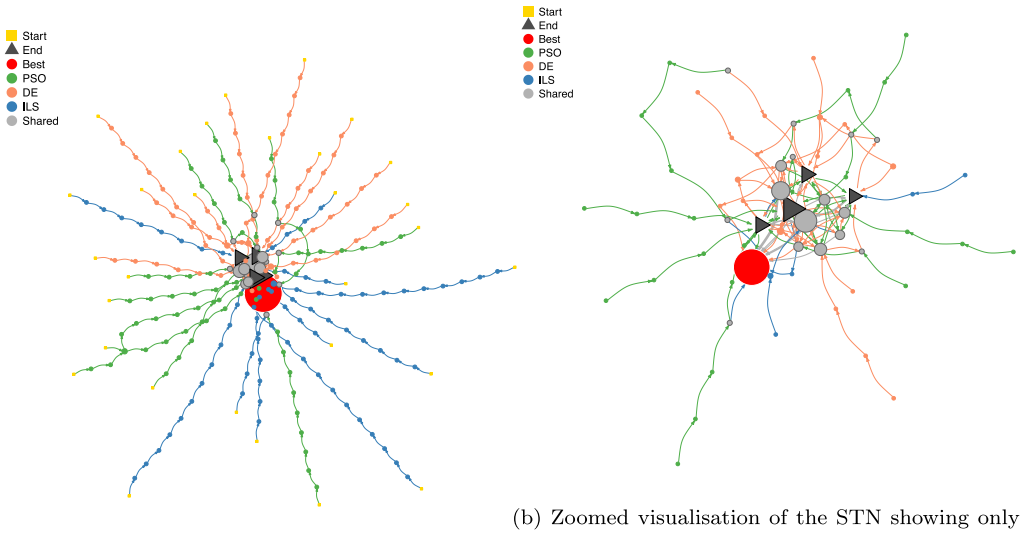
The production and visualisation of STNs in the case of the p-median problem is done using five problem instances from the related literature. In particular we make use of instances {pmed6, ..., pmed10} from OR-Library, which is a well-known collection of instances for a large number of problems.[2] All five problems have 200 demand points ($n = 200$) and request to open a varying number of facilities (that is, p ranges from 5 in pmed6 to 67 in pmed10).

5.3. Metaheuristic algorithms

We implemented an ant colony optimisation (ACO) approach, a biased random key genetic algorithm (BRKGA), and an iterated local search (ILS) metaheuristic for solving the p-median problem. Note that ACO and BRKGA are population-based approaches, while ILS is an algorithm based on local search. In all three cases we implemented rather standard algorithm versions, because the aim of this study is to demonstrate how the performance of the algorithms can be compared by means of STNs, rather than to develop high-performance versions of these algorithms. In the following we provide a brief description of the algorithm implementations.

Both ACO and BRKGA require a constructive procedure for generating feasible solutions. In the case of ACO, this procedure will be used for generating solutions at each iteration, while in the case of BRKGA, this procedure will be used for translating individuals into feasible solutions. In the following, let S with $|S| \leq p$ be a subset of the n demand points. Note that such a subset corresponds to a partial solution in case $|S| < p$, and

[2] These instances can be downloaded from http://people.brunel.ac.uk/~mastjjb/jeb/info.html.



(a) Full STN.

(b) Zoomed visualisation of the STN showing only those nodes within the first quantile of fitness values (the best 25% of the nodes).

Fig. 4. STNs for the Salomon benchmark.

to a complete solution otherwise. Let $j_{i,\min} := \text{argmin}_{j \in S} d_{ij}$ for all $i = 1, \ldots, n$. The objective function value $f(S)$ of a (partial) solution S can then be expressed as follows: $f(S) := \sum_{i=1}^{n} d_{i,j_{i,\min}}$. The basic procedure for constructing solutions to the p-median problem is shown in Algorithm 1. It starts with an empty solution $S = \emptyset$, and adds – at each iteration – exactly one demand point from $\{1, \ldots, n\} \setminus S$ to S until $|S| = p$. When used as a deterministic greedy algorithm, function Choose(S) (see line 3 of Algorithm 1) is implemented as follows. It chooses the demand point j^* such that $j^* := \text{argmin}_{j \in \{1,\ldots,n\} \setminus S} f(S \cup \{j\})$.

Implementation of ACO. We implemented a standard $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System in the Hyper-cube Framework as described, for example, in [48]. Only the pheromone model needs to be described, together with the way in which the pheromone values are used for constructing solutions at each algorithm iteration. More specifically, the algorithm makes use of a pheromone value $\tau_j \geq 0$ for each demand point $j = 1, \ldots, n$. When constructing a solution with Algorithm 1, function Choose(S) is implemented as follows. At each construction step, first, a probability $\mathbf{p}(j \mid S)$ for choosing $j \in \{1, \ldots, n\} \setminus S$ is determined as follows:

$$\mathbf{p}(j \mid S) := \frac{\tau_j / f(S \cup \{j\})}{\sum_{k \in \{1,\ldots,n\} \setminus S} \tau_k / f(S \cup \{k\})}$$

Second, a random value λ from $[0, 1]$ is chosen. If $\lambda \leq d_{\text{rate}}$, j^* is chosen such that $j^* := \text{argmax}_{j \in \{1,\ldots,n\} \setminus S} \mathbf{p}(j \mid S)$, that is, the demand point with the highest probability is deterministically chosen. Otherwise, j^* is determined by roulette wheel selection based on the probabilities. We chose a standard parameter value setting of 10 solution constructions per iteration, and a determinism rate (d_{rate}) of 0.7.

Implementation of BRKGA. The algorithm that was implemented for the p-median problem is a standard BRKGA as described in [49]. We only need to describe individuals and the procedure for producing a solution on the basis of an individual. More specifically, an individual is an array π of length n in which each position π_j ($j = 1, \ldots, n$) is a value from $[0, 1]$. The procedure of Algorithm 1 is used in the following way for translating an individual into a solution. At each step of Algorithm 1, function Choose(S) is implemented such that $j^* := \text{argmin}_{j \in \{1,\ldots,n\} \setminus S} \pi_j \cdot f(S \cup \{j\})$. For the experiments we used standard parameter values, such as a population size of 100, 15% of elite individuals, 20% of mutants, and a probability of inheritance from the elite parent of 0.6.

Implementation of ILS. The framework of an ILS (see, for example, [35]) is rather unsophisticated. The algorithm requires a starting solution, a mechanism for perturbing the current solution at each iteration, a local search procedure for improving the perturbed solutions, and an acceptance criterion for choosing the current solution for the next iteration. Our ILS implementation for the p-median problem uses a randomly generated solution as starting solution. The perturbation mechanism applies a series of x random demand point swaps, each one consisting of the removal of a random demand point from the current solution S_{cur} and adding a randomly chosen demand point from $\{1, \ldots, n\} \setminus S_{\text{cur}}$. The size of x is discussed below. Concerning the local search procedure, we use a first-improvement local search based on demand point swaps. Considering the demand points in the order given by their indices, the first improving swap that is encountered is performed. In order to shorten the running time of a single local search run, each local search run is stopped after at most $\lceil p/2 \rceil$ swaps. The acceptance criterion chooses either the current solution S_{cur}, or the solution obtained after applying perturbation and local search, to be the current solution of the next iteration. Our ILS simply chooses the best one among the two solutions. Finally, the number of perturbation swaps (x) is determined in a dynamic way that is known from the mechanism for choosing among different neighbourhoods in variable neighbourhood search (VNS) [50]. More specifically, x is taken from $\{\max\{2, \lfloor 0.05p \rfloor\}, \ldots, \lceil 0.5p \rceil\}$.

5.4. Search space partitioning

As shown before in the context of the continuous optimisation case study, the partitioning of the search space into locations containing a non-empty subset of solutions is an important tool for reducing the search trajectories of algorithms to their essential aspects. Remember that continuous search spaces were partitioned into discrete hypercubes of a predefined length, defining a location as a hypercube of solutions; see Section 4.4. However, due to the different nature of a combinatorial search space, search space partitioning cannot be done in the same way. In fact, while

the partitioning in a continuous search space is done independently from the generated search trajectories, in combinatorial spaces we found it more natural to apply a partitioning scheme that depends on the search trajectories. This scheme is outlined in the following.

Henceforth we assume that we are given a set $\mathcal{T}$ of search trajectories for the same problem instance, each one potentially produced by a different algorithm. Furthermore, we assume that these search trajectories refer to the original search space, that is, each one consists of a sequence of representative solutions. More precisely, each search trajectory $T \in \mathcal{T}$ is a sequence $s_1^T, \ldots, s_{|T|}^T$ of representative solutions. Moreover, we assume that each possible representative solution s is a string of characters $s[i]$, $i = 1, \ldots, |s|$. Hereby, each position i of s can be seen as a decision variable x_i with a domain D_i and $s[i] \in D_i$. In the case of the p-median problem, for example, a solution can be represented as a binary string of length n in which a position i (with domain $D_i = \{0, 1\}$) indicates whether or not a facility is located at demand point i. Given $\mathcal{T}$, let $\mathcal{S}(\mathcal{T})$ be the set of unique solutions contained in all the search trajectories of $\mathcal{T}$. Based on $\mathcal{S}(\mathcal{T})$ we can calculate the probability $\mathbf{p}(x_i = d)$ that domain value $d \in D_i$ appears at position $i = 1, \ldots, n$ of a solution:

$$\mathbf{p}(x_i = d) = \frac{|\{s \in \mathcal{S}(\mathcal{T}) \mid s[i] = d\}|}{|\mathcal{S}(\mathcal{T})|} \tag{2}$$

Intuitively, the less *variability* we find in the values taken by a decision variable (position) in the solutions of set $\mathcal{S}(\mathcal{T})$, the higher should be the chance to remove this variable from the search space for the purpose of partitioning, and vice versa. The variability in the values of a decision variable is a concept that is formally covered by a measure from information theory called *Shannon entropy* [51]. The Shannon entropy $H(x_i)$ of a discrete random variable x_i with domain D_i is formally defined as follows:

$$H(x_i) = -\sum_{d \in D_i} \mathbf{p}(x_i = d) \log_2 \mathbf{p}(x_i = d) \tag{3}$$

In particular, in the case of the lowest variability – that is, when $\mathbf{p}(x_i = d) = 1$ for some $d \in D_i$, and $\mathbf{p}(x_i = d') = 0$ for all $d' \neq d \in D_i$ – the Shannon entropy $H(x_i)$ evaluates to zero. On the contrary, in the case of the highest variability – that is, when $\mathbf{p}(x_i = d) = \mathbf{p}(x_i = d')$ for all $d, d' \in D_i$ – $H(x_i)$ evaluates to one. Consequently, we make use of the Shannon entropy (calculated on the basis of $\mathcal{S}(\mathcal{T})$) in order to produce a ranking L of all positions $i = 1, \ldots, n$. Note that the first entry in this list – that is, $L[1]$ – contains the position whose Shannon entropy is greater or equal to the one of all other positions. In order to partition the search space, we reduce this list to the first $z \leq n$ positions, resulting in a reduced list L_z. A location in the partitioned search space, obtained on the basis of L_z, contains all those solutions from the original search space that have the same value at all positions of L_z. Henceforth, let the reduction of a solution s to the positions in L_z be denoted by s_z. Then, two solutions s and s' from the original search space are mapped into the same location of the partitioned search space induced by L_z if and only if $s_z = s'_z$. Moreover, the objective function value $f(s_z)$ of s_z – in relation to set $\mathcal{S}(\mathcal{T})$ – is defined as follows: $f(s_z) := \min\{f(s') \mid s' \in \mathcal{S}(\mathcal{T}), s_z = s'_z\}$.[3] In other words, the objective function value of all solutions from $\mathcal{S}(\mathcal{T})$ that fall into the same location of the partitioned search space, is determined as the smallest objective function value of all these solutions.

Example 1 demonstrates combinatorial search space partitioning for a p-median problem with six demand points ($n = 6$) and the request to open $p = 3$ facilities. Moreover, the example applies a list L for search space partitioning of size $z = 2$, that is, the search space is partitioned based on two variables with the highest Shannon entropy value.

Example 1: Combinatorial Search Space Partitioning (p-median problem)

$$T_1 = \begin{pmatrix} s_1^{T_1} = \mathbf{101010} \\ s_2^{T_1} = \mathbf{001011} \\ s_3^{T_1} = \mathbf{011010} \\ s_4^{T_1} = \mathbf{010010} \end{pmatrix} \quad T_2 = \begin{pmatrix} s_1^{T_2} = \mathbf{111000} \\ s_2^{T_2} = \mathbf{100011} \\ s_3^{T_2} = \mathbf{011010} \end{pmatrix}$$

$$\mathcal{S}(\mathcal{T}) = \{s_1^{T_1}, s_2^{T_1}, s_3^{T_1}, s_4^{T_1}, s_1^{T_2}, s_2^{T_2}\}$$

Note that $s_3^{T_2}$ does not appear in $\mathcal{S}(\mathcal{T})$ because it is equal to $s_3^{T_1}$. As we deal with solutions in terms of binary strings, for each position $i = 1, \ldots, 6$ of a solution, a binary variable x_i is introduced. Based on the strings in $\mathcal{S}(\mathcal{T})$, the following probabilities for all domain values $\{0, 1\}$ can be calculated for each position.

$$\text{Probabilities: } \begin{pmatrix} \mathbf{p}(x_1 = 0) = 3/6 & \mathbf{p}(x_1 = 1) = 3/6 \\ \mathbf{p}(x_2 = 0) = 3/6 & \mathbf{p}(x_2 = 1) = 3/6 \\ \mathbf{p}(x_3 = 0) = 2/6 & \mathbf{p}(x_3 = 1) = 4/6 \\ \mathbf{p}(x_4 = 0) = 1 & \mathbf{p}(x_4 = 1) = 0 \\ \mathbf{p}(x_5 = 0) = 1/6 & \mathbf{p}(x_5 = 1) = 5/6 \\ \mathbf{p}(x_6 = 0) = 4/6 & \mathbf{p}(x_6 = 1) = 2/6 \end{pmatrix}$$

$$\text{Shannon entropy values: } \begin{pmatrix} H(x_1) = 1.0 \\ H(x_2) = 1.0 \\ H(x_3) = 0.918 \\ H(x_4) = 0.0 \\ H(x_5) = 0.650 \\ H(x_6) = 0.918 \end{pmatrix}$$

$L = (1, 2, 6, 3, 5, 4)$. The order between 1 and 2, and between 3 and 6, is randomly determined.
$L_{z=2} = (1, 2)$ (Example.: partitioning with $z = 2$)

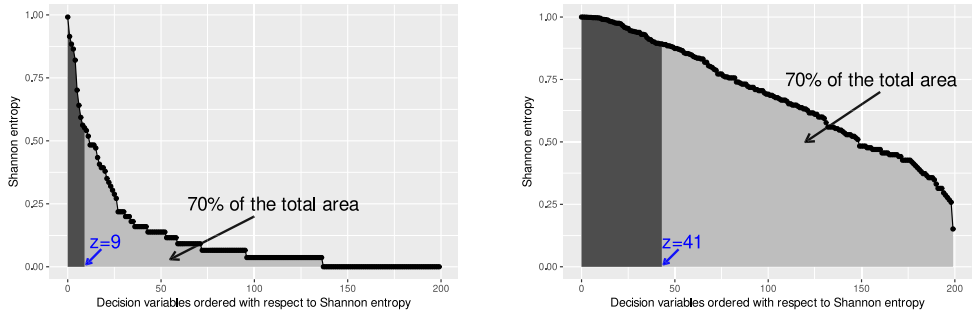
Trajectories in the partitioned search space:

$$T_1 = \begin{pmatrix} s_{1,z=2}^{T_1} = \mathbf{10} \\ s_{2,z=2}^{T_1} = \mathbf{00} \\ s_{3,z=2}^{T_1} = \mathbf{01} \\ s_{4,z=2}^{T_1} = \mathbf{01} \end{pmatrix} \quad T_2 = \begin{pmatrix} s_{1,z=2}^{T_2} = \mathbf{11} \\ s_{2,z=2}^{T_2} = \mathbf{10} \\ s_{3,z=2}^{T_2} = \mathbf{01} \end{pmatrix}$$

Note that $s_3^{T_1}$ and $s_4^{T_1}$, for example, are mapped to the same location in the partitioned search space, because $s_{3,z=2}^{T_1} = s_{4,z=2}^{T_1}$.

Note that an adequate value for $z > 0$ for the purpose of search space partitioning might not be easily found. Moreover, this value might be dependent on instance characteristics. In the context of the experimental evaluation we decided for the following scheme. The graphics in Fig. 5 contain plots of the Shannon entropy values of the variables in the order of L, that is, from left to right the Shannon entropy of the variables is non-increasing. As an example, this is done for two p-median instances with different characteristics: `pmed6` with ($n = 200, p = 5$), and `pmed10` ($n = 200, p = 67$). Both instances have 200 demand points (resulting in 200 variables) and a very different number of facilities to be opened. The value of $z > 0$ that corresponds to an $X\%$ **search space partitioning** is then determined as the largest integer value in $\{1, \ldots, n\}$ such that the area below the curve from the $z + t$-th variable to the last variable in L is at least $X\%$ of the total area below the curve. In Fig. 5(a) a 70% search space partitioning is obtained with $z = 9$ for instance

[3] This assumes a minimisation problem. In the case of maximisation one has to replace min by max.



(a) Problem instance pmed6. (b) Problem instance pmed10.

Fig. 5. Illustration of the way in which z-values for search space partitioning are derived based on the Shannon entropy values. The variables are ordered on the x-axis according to list L (from left to right). The two graphics show a 70% search space partitioning for two different problem instances.

Table 7
Performance metrics for ACO, BRKGA, and ILS: $\bar{f}$ (average objective function value) with standard deviation ($\pm\sigma$) and SRate (percentage of runs finding a global optimum.

Instance	ACO			BRKGA			ILS		
	$\bar{f}$	($\pm\sigma$)	SRate	$\bar{f}$	($\pm\sigma$)	SRate	$\bar{f}$	($\pm\sigma$)	SRate
pmed6	7824.0	(±0.0)	100%	7824.0	(±0.0)	100%	7824.0	(±0.0)	100%
pmed7	5631.0	(±0.0)	100%	5657.2	(±16.88)	10%	5631.0	(±0.0)	100%
pmed8	4450.9	(±9.42)	60%	4594.4	(±42.65)	0%	4445.0	(±0.0)	100%
pmed9	2753.9	(±2.13)	0%	2856.6	(±41.94)	0%	2734.0	(±0.0)	100%
pmed10	1286.7	(±12.09)	0%	1352.2	(±23.20)	0%	1255.0	(±0.0)	100%

Table 8
Merged STN metrics for the full (f) and the partitioned (p) search space.

	ntotal		*nbest*		*nend*		*nshared*		*best-strength*	
	f	p	f	p	f	p	f	p	f	p
pmed6	280	206	1	1	0	0	17	26	1.0	1.0
pmed7	423	312	2	2	9	9	5	13	0.7	0.8
pmed8	667	370	3	3	13	12	3	14	0.53	0.6
pmed9	816	243	10	2	20	16	0	11	0.33	0.43
pmed10	868	353	10	7	20	20	0	1	0.33	0.43

pmed6 and Fig. 5(b)) shows that the same partitioning is obtained for pmed10 with $z = 41$. In other words, our scheme adapts to instance/algorithm characteristics. In the case of pmed6, all trajectories that were used to produce these Shannon entropy values quickly focus on a certain area of the search space. For this reason there are many variables with very low Shannon entropy values. This is very different in the case of the search trajectories used for producing Fig. 5(b), which shows many variables with rather high Shannon entropy values. In this second case, it would certainly not make sense to produce a search space partitioning based on very few variables. Finally, note that a 0% search space partitioning corresponds to not applying any search space partitioning.

5.5. Experimental results

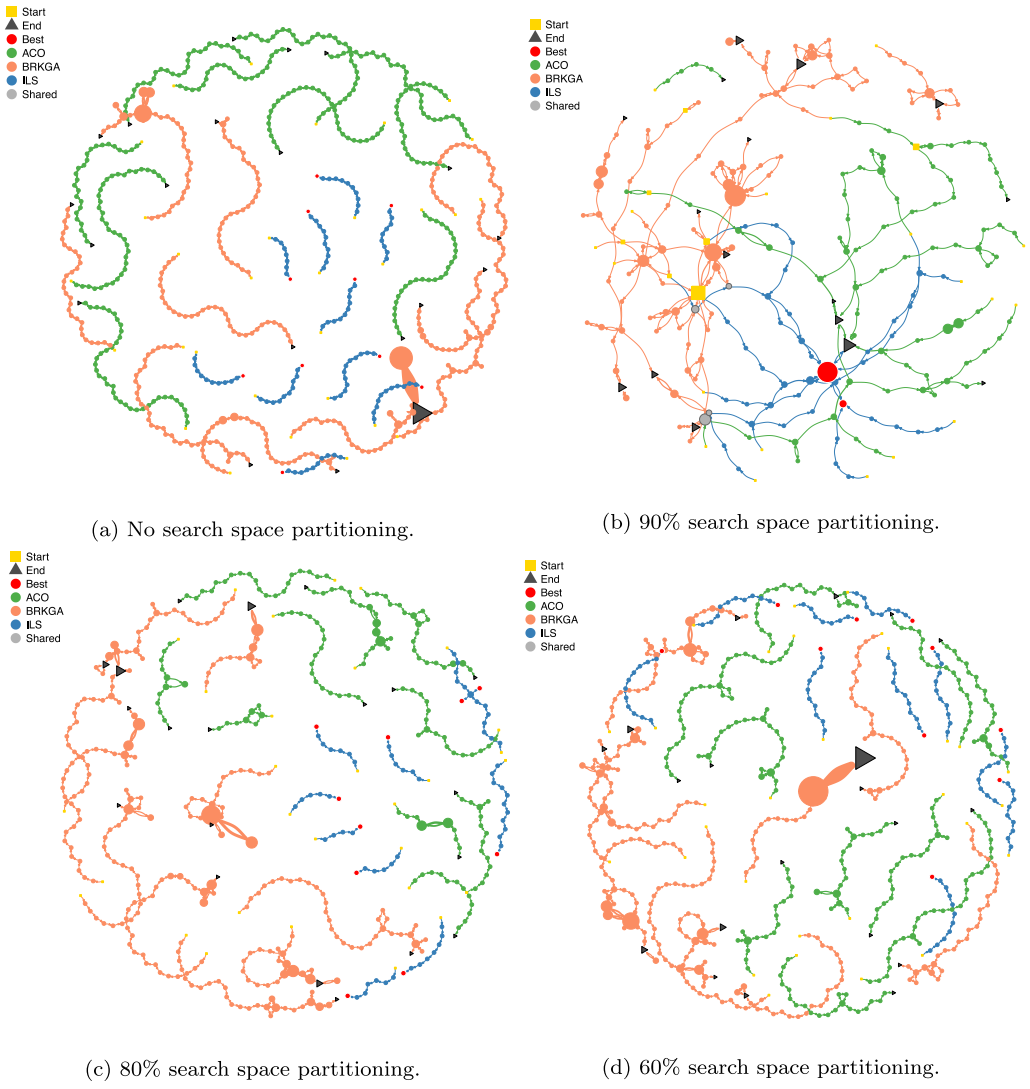
The experimental setup in the context of this combinatorial case study is as follows. After implementing the three algorithms described before, each one was applied 10 times to the five problem instances. The time limit for each run was set to 100 CPU seconds. Table 7 provides the obtained performance metrics. In particular, for each pair of a problem instance and an algorithm the table provides the average objective function value obtained over 10 runs ($\bar{f}$), together with the corresponding standard deviation ($\pm\sigma$), and the success rate, that is, the percentage of runs (out of 10) that ended up in a global optimum.

Results and Discussion. From a global perspective it can be observed that the problem difficulty seems to increase steadily from pmed6 (easiest) to pmed10 (hardest). This is shown by the decreasing success rates of both ACO and BRKGA. ILS is clearly the best-performing algorithm with a success rate of 100% for all five instances. In addition to the performance metrics, the values of the five metrics described in Section 3.3 were calculated on the basis of the five merged STNs (one per problem instance). The values of these metrics can be found in Table 8. In the following we will interpret the results based on three sources of information: the performance metrics, the STN metrics, and visualisations of the merged STNs.

Fig. 6 shows merged STNs for p-median instance pmed9, which has a p-value of 40. This means that the Shannon entropy value distribution of the variables obtained by the algorithm trajectories – calculated on the basis of solution set $\mathcal{S}(\mathcal{T})$ – can be expected to resemble the one from Fig. 5(b). We decided to show both the original STN – that is, without search space partitioning – as well as the same STN for adequate and inadequate partitioning percentages.[4] Fig. 6(a) displays the original STN for pmed9, while the STN from Fig. 6(b) results from an adequate search space partitioning of 90%, obtained by a limitation of the search space to the $z = 11$ variables with the highest Shannon entropy values. The remaining two STNs in this figure show that search space partitioning of 60%, respectively 80%, are not sufficient yet in order to be able to interpret the STN. Note that the graphics in Figs. 6(a), 6(c) and 6(d) are produced with the Kamada–Kawai (KK) layout [33], because this layout is better for separating disconnected components. The following can be observed:

- The original (full-size) STN (Fig. 6(a)) does not show any overlap between the algorithm trajectories, which holds both for trajectories from the same algorithm and for trajectories from different algorithms. This is verified by the value of metric *nshared* in Table 8. In fact, the 10 ILS runs converge to 10 different optimal solutions. In contrast, the STN visualised after a search space partitioning of 90% (see

[4] Remember that in the continuous optimisation case study, the notion of an *original STN* is not applicable, as STNs must necessarily be displayed in a partitioned search space.



(a) No search space partitioning.

(b) 90% search space partitioning.

(c) 80% search space partitioning.

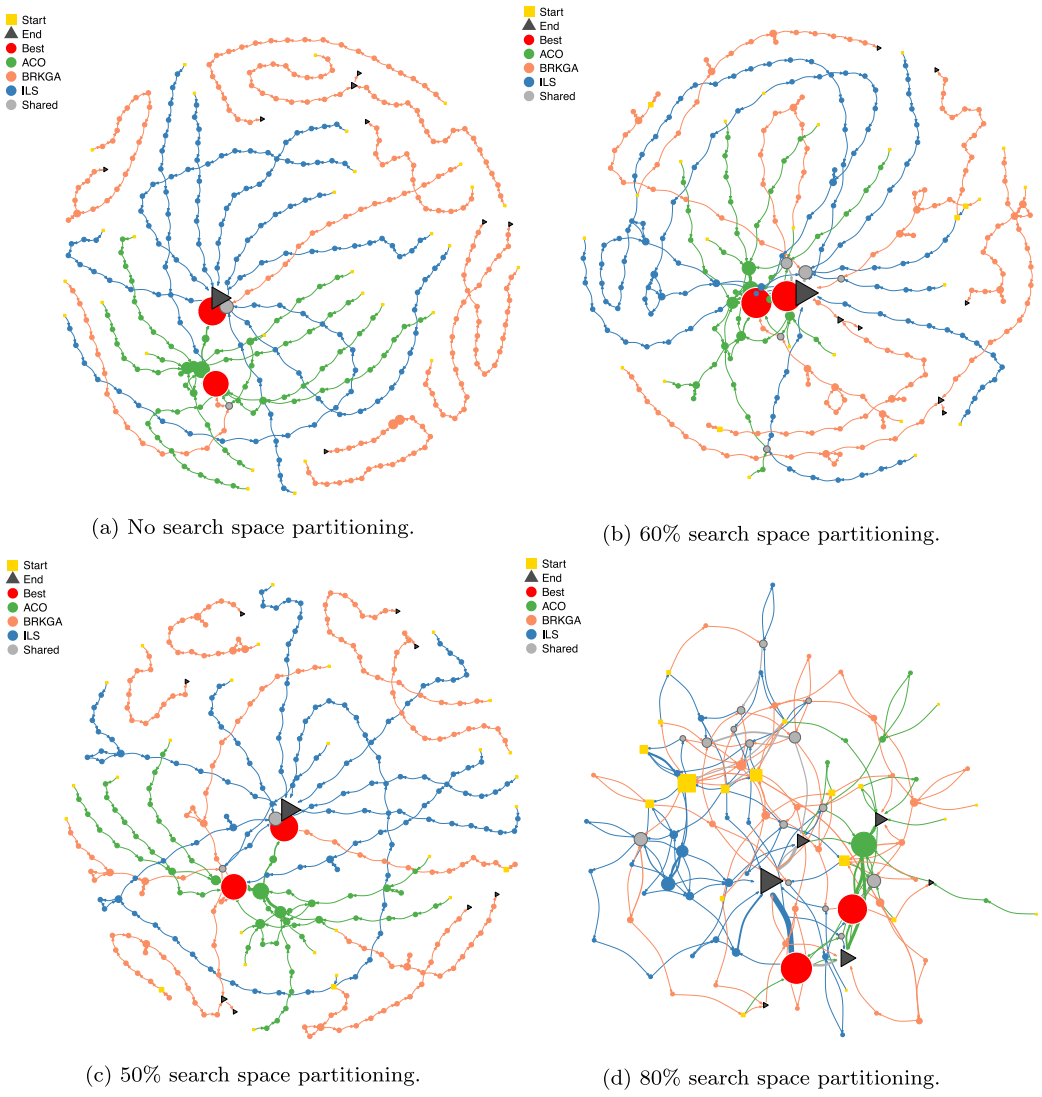
(d) 60% search space partitioning.

Fig. 6. Merged STNs for instance pmed9.

Fig. 6(b)) shows that all runs of ILS are attracted by the same area of the search space. In fact, all ILS runs end up in the same best-found location. This is also indicated by the value 1.0 of metric *best-strength* for the individual ILS STN in Fig. 8. Furthermore, many runs of BRKGA are now, after search space partitioning, interconnected. The same holds for ACO. In fact, observe the value 11 of metric *nshared* in Table 8, in comparison to value zero in the case of no search space partitioning. As a side-comment, the two larger orange and dark grey dots in the lower, right part of the full-size STN show an oscillation or cycling behaviour of BRKGA between two similar solutions of the same quality.

- In the upper part of Fig. 6(b) we can see several self-cycles in the BRKGA trajectories. Even though visiting more solutions than the other two approaches, as indicated by the individual *nodes* metric for pmed9 in Fig. 8, this indicates that BRKGA quickly gets trapped in different areas of the search space. This also indicates that the length of a single step in the full-size search space is rather small and that BRKGA tends to converge to solutions rather close to the initial solutions.
- Concerning inter-algorithm overlap, the STN after search space partitioning (Fig. 6(b)) shows that runs of ILS and BRKGA have some regional overlap during early stages of the search process, while ILS and ACO show some regional overlap in later stages of the search process (see the largest dark-grey triangle close to the large red dot). Note that we refer to *regional overlap*, because dots, squares and triangles in Figs. 6(b)–6(d) correspond to locations in the search space, rather than to single solutions.
- The best solutions are found by ILS (see the red dots). In particular, the STN without search space partitioning shows that ILS finds a different solution of the same quality in all 10 runs. Moreover, from the success rate of ILS in Table 7 we know that all these solutions are optimal. Nevertheless, the STN after search space partitioning shows that these solutions are very similar, because 9 final solutions (the ends of 9 ILS trajectories) are merged into only one single location of the partitioned search space. This is also confirmed by metric *nbest* in Table 8.

As second example we consider a problem instance with different characteristics. Fig. 7 displays four STNs for p-median instance pmed7, which has a p-value of 10. The distribution of the Shannon entropy values of the variables can therefore be expected to resemble the one from Fig. 5(a). The STN in Fig. 7(a) is the original STN, the one in Fig. 7(b) is obtained after a search space partitioning of 60%, while the remaining two are obtained after search space partitioning of 50% (too few) and 80% (too



(a) No search space partitioning.

(b) 60% search space partitioning.

(c) 50% search space partitioning.

(d) 80% search space partitioning.

Fig. 7. STNs for instance pmed7.

much), respectively. Table 7 shows that both ACO and ILS have a success rate of 100% for this problem instance. Moreover, one run of BRKGA (success rate of 10%) converges to an optimal solution. Metric *nbest* from Table 8 indicates that altogether two different optimal solutions are found. Moreover, value 9 of metric *nend* indicates that all the 9 BRKGA runs that do not find an optimal solution converge to different solutions. One of them gets actually stuck in the large dark-grey rectangle through which 9 out of 10 ILS runs pass in order to reach an optimal solution. Note that the adequate search space partitioning of 60% results in a reduction from an initial number of 200 variables to the $z = 19$ variables with the highest Shannon entropy values. The following can be observed when comparing the STN graphics between each other and also when contrasting them with the two graphics obtained for problem instance pmed9:

- First of all, the STN without search space partitioning already shows some overlap between the trajectories of different algorithms, as indicated by the light grey dots and the dark-grey triagle close to the two red dots. This observation is confirmed by the value 5 of metric *nshared* in Table 8.
- In comparison to the results obtained for instance pmed7, BRKGA seems to work better for instance pmed9. There is one BRKGA trajectory that reaches one of the two optimal solutions (red dots).
- Interestingly, the solutions to which the 30 algorithm runs converge are so different from each other that they are still mapped to different locations in the partitioned search space. This is indicated by the values of metrics *nbest* and *nend* in Table 8. In particular, the values do not change from the original STN to the STN after search space partitioning.
- As already mentioned above, the algorithms found two optimal solutions (red dots). Interestingly, the STN when displayed in the partitioned search space (Fig. 6(b)) shows that the left one of the two is mostly found by the ACO runs, while the other one is mostly found by the ILS runs. In other words, the two algorithms are attracted to different optimal solutions of the same quality.
- The STN when shown in the partitioned search space (Fig. 7(b)) shows regional overlap especially between the BRKGA and the ILS runs. However, it becomes clear that BRKGA, most of the time, converges before reaching solutions of the highest quality. In particular, there is one BRKGA run that converges to the large dark-grey triangle close to the optimal solutions that is an attractor for the ILS runs.

6. Conclusion

We proposed *search trajectory networks* (STNs), a network-based model to characterise and visualise the search behaviour

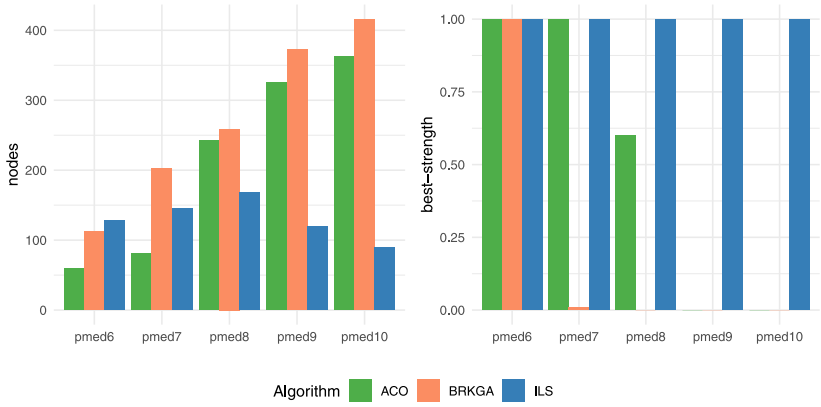


Fig. 8. Algorithm specific STN metrics for the p-median instances. A description of the metrics is found in Table 1.

of metaheuristics. We showed that STNs can be applied to algorithms from the main metaheuristic paradigms: stochastic local search, evolutionary algorithms and swarm intelligence, and to both continuous and combinatorial optimisation. We argue, therefore, that STNs can be applied to analyse any metaheuristc and problem domain. One strength of our approach is that it does not require additional methods for sampling the search process, instead, the data to build the network models is collected from a number of runs of the algorithms under study. Our analysis illustrates that the qualitative (visualisations) and quantitative (network metrics) analysis of STNs give interesting insight into the convergence behaviour of algorithms and their performance differences. STNs allow us to observe and quantify which portions of the search space attract the process and thus act as traps in the way of locating the best solution. We can also identify frequently traversed areas of the search space by a given algorithm or set of algorithms, as well as the existence of cycling (oscillating) behaviour. We argue that this information gives new insights into understanding the dynamics of metaheuristics, and thus can be used to improve their design and to inform the selection of the most suitable algorithm for a given problem. By providing the source code for constructing, visualising and analysing the network models, we hope to provide an accessible new tool for the analysis and comparison of metaheuristic algorithms.

Future work will analyse real-world optimisation problems as well as scenarios where significant performance differences among algorithms are known to exist but are not well understood. We will also study the impact on the trajectories of considering alternative search operators, as well as the relationship between the search trajectories and the fitness landscape structure of the underlying optimisation problem. We argue that our proposed approach will shed new light into these scenarios, which will have implications for algorithm selection and understanding search difficulty.

CRediT authorship contribution statement

Gabriela Ochoa: Conceptualization, Methodology, Software, Validation, Data curation, Investigation, Writing - original draft, Writing - review & editing. **Katherine M. Malan:** Conceptualization, Methodology, Software, Validation, Investigation, Writing - original draft, Writing - review & editing. **Christian Blum:** Conceptualization, Methodology, Software, Validation, Investigation, Writing - original draft, Writing - review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

Christian Blum was funded by project CI-SUSTAIN of the Spanish Ministry of Science and Innovation (PID2019-104156GB-I00). Katherine Malan was funded by the National Research Foundation of South Africa (Grant Number: 120837).

References

[1] K. Sörensen, Metaheuristics-the metaphor exposed, Int. Trans. Oper. Res. 22 (2013) 3–18.
[2] P. Calégari, G. Coray, A. Hertz, D. Kobler, P. Kuonen, A taxonomy of evolutionary algorithms in combinatorial optimization, J. Heuristics 5 (1999) 145–158.
[3] M.A. Lones, Mitigating metaphors: A comprehensible guide to recent nature-inspired algorithms, SN Comput. Sci. 1 (2019).
[4] C. Blum, A. Roli, Metaheuristics in combinatorial optimization, ACM Comput. Surv. 35 (2003) 268–308.
[5] A.E. Eiben, C.A. Schippers, On evolutionary exploration and exploitation, Fund. Inform. 35 (1998) 35–50.
[6] T.D. Collins, Applying software visualization technology to support the use of evolutionary algorithms, J. Vis. Lang. Comput. 14 (2003) 123–150.
[7] H. Pohlheim, Multidimensional scaling for evolutionary algorithms – visualization of the path through search space and solution space using Sammon mapping, Artif. Life 12 (2006) 203–209.
[8] K. Michalak, Low-dimensional euclidean embedding for visualization of search spaces in combinatorial optimization, IEEE Trans. Evol. Comput. 23 (2019) 232–246.
[9] A.D. Lorenzo, E. Medvet, T. Tušar, A. Bartoli, An analysis of dimensionality reduction techniques for visualizing evolution, in: Proceedings of the Genetic and Evolutionary Computation Conference Companion, ACM, 2019.
[10] M.E.J. Newman, The structure and function of complex networks, SIAM Rev. 45 (2003) 167–256.
[11] M.E.J. Newman, Networks: An Introduction, Oxford University Press, Oxford, New York, 2010.
[12] G. Ochoa, K.M. Malan, C. Blum, Search trajectory networks of population-based algorithms in continuous spaces, in: Applications of Evolutionary Computation - 23rd European Conference, EvoApplications 2020, in: Lecture Notes in Computer Science, vol. 12104, Springer, 2020, pp. 70–85.
[13] G. Ochoa, M. Tomassini, S. Verel, C. Darabos, A study of nk landscapes' basins and local optima networks, in: Genetic and Evolutionary Computation Conference, GECCO, ACM, 2008, pp. 555–562.
[14] S. Verel, G. Ochoa, M. Tomassini, Local optima networks of NK landscapes with neutrality, IEEE Trans. Evol. Comput. 15 (2011) 783–797.
[15] J.P.K. Doye, The network topology of a potential energy landscape: a static scale-free network, Phys. Rev. Lett. 88 (2002) 238701.
[16] O.M. Becker, M. Karplus, The topology of multidimensional potential energy surfaces: Theory and application to peptide structure and kinetics, J. Chem. Phys. 106 (1997) 1495.
[17] J.P.K. Doye, M.A. Miller, D.J. Wales, The double-funnel energy landscape of the 38-atom Lennard-Jones cluster, J. Chem. Phys. 110 (1999) 6896–6906.
[18] C. Flamm, I.L. Hofacker, P.F. Stadler, M.T. Wolfinger, Barrier trees of degenerate landscapes, Phys. Chem. 216 (2002) 155–173.
[19] J. Hallam, A. Prugel-Bennett, Large barrier trees for studying search, IEEE Trans. Evol. Comput. 9 (2005) 385–397.
[20] I. Zelinka, D. Davendra, Investigation on relations between complex networks and evolutionary algorithm dynamics, Int. J. Comput. Inf. Syst. Ind. Manag. Appl. 3 (2011) 236–247.

[21] P. Gajdo, P. Kromer, I. Zelinka, Network visualization of population dynamics in the differential evolution, in: IEEE Symposium Series on Computational Intelligence, pp. 1522–1528.
[22] L. Skanderová, T. Fabian, I. Zelinka, Small-world hidden in differential evolution, in: IEEE Congress on Evolutionary Computation, CEC, pp. 3354–3361.
[23] M. Oliveira, C.J.A. Bastos-Filho, R. Menezes, Towards a network-based approach to analyze particle swarm optimizers, in: 2014 IEEE Symposium on Swarm Intelligence, IEEE, 2014.
[24] L. Taw, N. Gurrapadi, M. Macedo, M. Oliveira, D. Pinheiro, C. Bastos-Filho, R. Menezes, Characterizing the social interactions in the artificial bee colony algorithm, in: 2019 IEEE Congress on Evolutionary Computation, CEC, IEEE, 2019.
[25] P. Bosman, A.P. Engelbrecht, Diversity rate of change measurement for particle swarm optimisers, in: Swarm Intelligence, ANTS 2014, in: LNCS, vol. 8667, Springer International Publishing, 2014, pp. 86–97.
[26] O. Olorunda, A.P. Engelbrecht, Measuring exploration/exploitation in particle swarms using swarm diversity, in: 2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence), IEEE, 2008.
[27] T. Tusar, B. Filipic, Visualization of pareto front approximations in evolutionary multiobjective optimization: A critical review and the prosection method, IEEE Trans. Evol. Comput. 19 (2015) 225–245.
[28] J.E. Fieldsend, T. Chugh, R. Allmendinger, K. Miettinen, A feature rich distance-based many-objective visualisable test problem generator, in: Proceedings of the Genetic and Evolutionary Computation Conference, ACM, 2019.
[29] S. Herrmann, G. Ochoa, F. Rothlauf, Pagerank centrality for performance prediction: the impact of the local optima network model, J. Heuristics 24 (2018) 243–264.
[30] G. Ochoa, N. Veerapen, Mapping the global structure of TSP fitness landscapes, J. Heuristics 24 (2018) 265–294.
[31] G. Csardi, T. Nepusz, The igraph software package for complex network research, InterJournal Complex Syst. (2006) 1695.
[32] T.M.J. Fruchterman, E.M. Reingold, Graph drawing by force-directed placement, Softw. Pract. Exper. 21 (1991) 1129–1164.
[33] T. Kamada, S. Kawai, An algorithm for drawing general undirected graphs, Inform. Process. Lett. 31 (1989) 7–15.
[34] A.M. Sutton, D. Whitley, M. Lunacek, A. Howe, PSO and multi-funnel landscapes: how cooperation might limit exploration, in: Proceedings of the 8th Annual Genetic and Evolutionary Computation Conference, pp. 75–82.
[35] H. Ramalhinho Lourenço, O.C. Martin, T. Stützle, Iterated Local Search: Framework and Applications, Springer International Publishing, pp. 129–168.
[36] K.V. Price, R.M. Storn, J.A. Lampinen, Appendix A.1: Unconstrained unimodal test functions, in: Differential Evolution a Practical Approach to Global Optimization, in: Natural Computing Series, Springer-Verlag, Berlin, Germany, 2005, pp. 514–533.
[37] X. Yao, Y. Liu, G. Lin, Evolutionary programming made faster, IEEE Trans. Evol. Comput. 3 (1999) 82–102.
[38] S.K. Mishra, Performance of Repulsive Particle Swarm Method in Global Optimization of Some Important Test Functions: A Fortran Program, Technical Report, 2006, Social Science Research Network (SSRN).
[39] R. Eberhart, J. Kennedy, A New Optimizer using Particle Swarm Theory, in: Proceedings of the Sixth International Symposium on Micromachine and Human Science, pp. 39–43.
[40] J. Kennedy, R. Eberhart, Particle Swarm Optimization, in: Proceedings of the IEEE International Joint Conference on Neural Networks, pp. 1942–1948.
[41] Y. Shi, R. Eberhart, A modified particle swarm optimizer, in: Proceedings of the 1998 IEEE World Congress on Computational Intelligence, 1998, pp. 69–73.
[42] R. Eberhart, Y. Shi, Comparing Inertia Weights and Constriction Factors in Particle Swarm Optimization, in: Proceedings of the IEEE Congress on Evolutionary Computation, Vol. 1, pp. 84–88.
[43] R. Storn, K. Price, Minimizing the real functions of the ICEC'96 contest by differential evolution, in: Proceedings of the International Conference on Evolutionary Computation, pp. 842–844.
[44] D.J. Wales, J.P.K. Doye, Global optimization by basin-hopping and the lowest energy structures of lennard-jones clusters containing up to 110 atoms, J. Phys. Chem. A 101 (1997) 5111–5116.
[45] D. Liu, J. Nocedal, On the limited memory bfgs method for large scale optimization, Math. Program. 45 (1989) 503–528.
[46] P. Virtanen, R. Gommers, T.E. Oliphant, et al., SciPy 1.0: fundamental algorithms for scientific computing in Python, Nature Methods 17 (2020) 261–272.
[47] M.T. Melo, S. Nickel, F. Saldanha-Da-Gama, Facility location and supply chain management–a review, European J. Oper. Res. 196 (2009) 401–412.
[48] C. Blum, M. Dorigo, The hyper-cube framework for ant colony optimization, IEEE Trans. Syst. Man Cybern. B 34 (2004) 1161–1172.
[49] J.F. Gonçalves, M.G.C. Resende, Biased random-key genetic algorithms for combinatorial optimization, J. Heuristics 17 (2011) 487–525.
[50] P. Hansen, N. Mladenović, J. Brimberg, J.A. Moreno Pérez, Variable neighborhood search, Springer International Publishing, pp. 57–97.
[51] C.E. Shannon, A mathematical theory of communication, Bell Syst. Tech. J. 27 (1948) 379–423.