Designing and Tuning SLS through Animation and Graphics – an extended walk-through

Steven Halim and Roland H.C. Yap

School of Computing, National University of Singapore
{stevenha,ryap}@comp.nus.edu.sg

Abstract. Stochastic Local Search (SLS) is quite effective for a variety of Combinatorial (Optimization) Problems (COP). However, the performance of SLS depends on several factors and getting it right is not trivial. In practice, SLS may have to be carefully designed and tuned to give good results. Often this is done in an ad-hoc fashion. One approach to this issue is black-box where a tuning algorithm is used to find good parameters for the black-box SLS. Another approach is white-box which takes advantage of the human in the process. In this paper, we show how visualization using a generic visual tool can be effective for a white-box approach to get the right SLS behavior on the fitness landscape of the problem instances at hand. We illustrate this by means of an extended walk-through on the Quadratic Assignment Problem. At the same time, we present the human-centric tool which have been developed.

1 Introduction

Stochastic Local Search (SLS) algorithms have been used to attack various NP-hard Combinatorial (Optimization) Problems (COP), often with satisfactory results. However, practitioners usually encounter difficulties when they try to get good performance from an SLS implementation.

Often the COPs encountered in industry are new COPs or variants of the classical COPs where no or little research is available. Sometimes a COP C' resembles a *classic* COP C for which a good SLS S is known. However, a direct implementation of that SLS S into the actual COP C' usually will not immediately yield good performance. As there is no universal SLS which has good performance on all COPs, it is necessary to adapt an existing SLS or create a new SLS for the COP at hand. It is often said that it is easy to create a working SLS for a COP, but hard to tune the SLS to achieve good performance on various instances of the COP [1–5]. [4] has summarized this situation as — 10% of the development time is spent in designing, implementing, and testing the SLS while the remaining 90% is for (fine) tuning the SLS.

In several papers, SLS tuning is focused on tuning the parameter values for the SLS. This, however, does not tell the whole story. From the beginning, one would already have a large number of combinations of SLS components (e.g. the neighborhood, etc) and search strategies (e.g. intensification, diversification, etc) to configure the SLS. These can also determine the performance of the SLS. Here we consider the SLS *design and tuning* problem as a *holistic* problem of finding a suitable configuration including parameter values, choice of components, and search strategies for an SLS in order to give good results for the class of COP instances at hand within *limited development time* and *limited running time*. This definition takes into account the practical efforts and resource requirements of industrial problems.

In [5], we classified the existing works on SLS design and tuning problem into two major approaches. Both approaches require the interaction of the human (the SLS algorithm designer) with the computer but in different ways:

1. The *black-box approach* favors the utilization of computer to do fine-tuning. Black-box approach aims to develop special 'tuning algorithms' to explore the SLS configuration space initially given by the algorithm designer. The best found configuration among the *given* configuration space is returned by the tuning algorithm. Examples of black-box approaches include F-Race [2], CALIBRA [4], +CARPS [6], Search Parameter Optimization [7], etc.
2. The *white-box approach* favors the utilization of human intelligence and/or visual perception strength. White-box approach aims to create tools or methods to help analyzing the performance of the SLS so that the algorithm designer has a basis for tweaking the SLS implementation. In this fashion, the SLS may be redesigned to extend beyond the initial configuration space. Examples of white-box approaches include Statistical Analysis (Fitness Distance Correlation Analysis, Run Time Distribution, etc) [3, 8], Sequential Parameter Optimization [9], Human Guided Tabu Search [10], Visualization of Search Process [11, 12], V-MDF and Viz [5, 13–15], etc.

In this paper, we present a white-box approach where the algorithm designer first 'opens the white-box' using intuitive visualization tools. The objective is to investigate the *fitness landscape* [3, 8, 16] of the COP instance at hand and the SLS *trajectory* on the fitness landscape. This allows the algorithm designer to *understand* the kinds of problems which might be affecting the SLS, e.g. stuck in a deep local optimum of a rugged fitness landscape, failure to navigate to good regions, etc. The algorithm designer can use that knowledge to *roughly tune* the SLS either by using *known* or *new* heuristic tweaks. This process may involve not only changing parameter values but also the SLS algorithm design.

We show how visualization can be effective for designing and tuning an SLS algorithm in a white-box fashion. We have chosen as a starting baseline, a Robust Tabu Search (Ro-TS) algorithm which has been shown to give good performance for the Quadratic Assignment Problem (QAP) [17]. We show how by using *generic* visualization tools as an integral part of the development process, one can get insights into the fitness landscapes and problem characteristics of the instances of a COP. In turn, the insights inspire the development of an improved SLS that is suitable to navigate the fitness landscape.

Note: Several aspects of the presentation in this paper are best viewed via the animated video at this URL: `http://www.comp.nus.edu.sg/~stevenha/viz`.

Here we have tried to illustrate the main points of our approach under the constraints imposed by a static, black and white paper (the online version of this paper in the Springer website is in color and can be magnified).

2 The Visualization Tool: Viz

We developed a visualization tool for white-box approach called Viz. Viz currently consists of: (a) the Viz Experiment Wizard (EW); and (b) the Viz Single Instance Multiple Runs Analyzer (SIMRA).

The initial idea of using anchor points to explain SLS trajectory was first proposed in [5, 13]. The proposal of a generic abstract 2-D visualization of COP fitness landscape using anchor points and visualization of SLS trajectory on it was proposed in the preliminary version of Viz system [14]. The GUI and animation aspects of the visualizations in Viz are detailed in [15]. We remark that the Viz system is still undergoing development. As such, the visualizations and GUI in this paper are significantly improved over [14, 15].

Basic Concepts
Given a COP instance π, the *fitness landscape* of π, $FL(\pi)$, is defined[1] as $FL(\pi) : (S(\pi), d(s_1, s_2), g(\pi))$ [8], where $S(\pi)$ is the set of solutions of the COP instance (the search space), $d(s_1, s_2) : S \times S \rightarrow \Re$ is a distance metric which is natural for the COP in question (e.g. Hamming distance for QAP, see also the discussion in [18]), and $g(\pi) : S \rightarrow \Re$ is the objective function. One can visualize the abstract 2-D fitness landscape as a surface where each solution is a point on the surface, separated from other points according to its distance to those other points, and with a height that reflects the fitness (objective value) of that solution.

The *search trajectory* of an SLS algorithm on this $FL(\pi)$ is defined as a finite sequence $(s_0, s_1, \ldots, s_k) \in S(\pi)$ of search positions corresponding to solutions in $S(\pi)$ (a solution can be satisfiable or not satisfiable) such that $\forall i \in \{1, 2, \ldots, k\}$, (s_{i-1}, s_i) is a local move done by the SLS according to its neighborhood $N(\pi)$ [3]. The SLS algorithm can be visualized as a heuristic algorithm that navigates or walks through the fitness landscape of the COP instance in order to find higher (lower for minimizing COP) peaks (valleys) of the fitness landscape.

A Generic Fitness Landscape and Search Trajectory Visualization
[14, 15] proposed that observing search trajectory on a fitness landscape is helpful to understand what is happening in the search. We describe the generic Fitness Landscape and Search Trajectory (FLST) visualization in Viz below:

First, we introduce the concept of Anchor Point (AP) set. The fitness landscape can contain an exponential number of solutions, thus we need an AP set which represents selected reference points on the fitness landscape to form an *approximate* fitness landscape visualization. The points along the SLS trajectory are then replayed back according to their distance w.r.t APs in the AP set.

[1] The definition that we use here [8] depends on the distance metric $d(s_1, s_2)$ and not the neighborhood $N(\pi)$ as in [3]. Thus, changing the $N(\pi)$ of the SLS will not change the fitness landscape but will change the SLS trajectory on that fitness landscape.

In overview, FLST visualization is currently prepared using the following steps:

1. Potential AP Collection Phase (see Fig 1)
In order to use FLST visualization in Viz, an SLS algorithm is augmented to save a history of the solutions visited into a log (i.e. RunLog file). In an SLS trajectory, the solution just before a non-improving move occurs is likely to be a local optimum solution w.r.t to the neighborhood used by the SLS. We use this feature to collect local optima solutions recorded in the RunLog files from various SLS runs. These local optima solutions form a potential AP set.

2. AP Set Update Phase (see Fig 1)
For visualization, it is not feasible to display too many points on screen. Thus, the AP set size should be limited to a small fraction of the possible solution set. It is important to only select meaningful APs that have these criteria:

1. Diverse: if AP X is included, then a similar AP X' will not.
2. High quality: good local optima must be included if found.
3. Important (best found, often visited) solutions from each run are included.

To achieve the above-mentioned criteria, we update the AP set according to the strategy below. The quality of the APs in the AP set is improved over time while the diversity of the APs is roughly maintained as we do more SLS runs:

1. Load past AP set from an AP log file if it exists.
2. If the AP set is still not full, *randomly* add any potential AP into it until it is full. The random addition of potential AP encourages diversity.
3. If the AP set is already full, compare each potential AP P with all current APs in AP set. Pick the nearest AP Q in term of distance metric w.r.t P. Then, if P is better than Q, P will replace Q. Otherwise, ignore P.
4. Save the current AP set into an AP log file for future experiments.

3. AP Layout Phase (see Fig 2)
Now that we have a set of n-dimensional APs, how to visualize it? Our strategy is to use an abstract 2-D fitness landscape visualization which is guided by the distance metric between any two AP points. We use a spring model layout algorithm – which is similar to a statistical multi-dimensional scaling technique. Any two APs are connected by a virtual spring which its natural length equal to the distance between the two APs. The spring model will layout the points such that points that are near in abstract 2-D fitness landscape are *approximately* near in the actual n-dimensional space and vice versa. The current implementation uses NEATO algorithm from Graphviz [19] to do the AP layout.

4. AP Labeling Phase (see Fig 3)
Next, we label the APs based on their quality using the AP legend in Fig 3. We use both color and shape to label the quality of the APs because even though color alone is easily seen on the screen, it is hard to see in black and white print. Newly added APs from recent runs are highlighted with a pink border so that the positions and qualities of the new APs can be easily identified.

The purpose of AP layout and labeling is to identify patterns in the fitness landscape at a glance, e.g. the distribution (clustered, spread, etc) and the quality (low variance ≈ smooth, high variance ≈ rugged, etc) of the old and new anchor points. This visualization extends the well known FDC scatter plot as it provides more information at a glance. It does not only measure distance and fitness w.r.t best found solution but *within all* APs in the AP set.

In Fig 7, we show an alternative side view of the abstract 2-D fitness landscape that emphasizes the smoothness or ruggedness of the APs.

5. Search Trajectory Layout Phase (see Fig 4)
Finally, we measure the distance between each point of the SLS run w.r.t to all points in the selected AP set. When the current point in the SLS run is near (by the distance metric) to one or more AP (this is possible since the points are n-dimensional), we draw a circle on those APs indicating that the current point is near those APs. Otherwise, we draw an indicator (see the example of a radar-like box in Fig 10, left) to tell how far is the current point to the nearest AP. This *near-ness factor* is adjustable, ranges from 0 (exact match) to n (total mismatch), and is visualized via the radius of the enclosing circle. The drawing of the search trajectory as a *trail* is done over time using animation to indicate how the search progresses.

In Fig 4, we describe some possible interpretations of the search trajectories which are based on the four runs in Fig 1 and the fitness landscape in Fig 2 and Fig 3. As Fig 4 is static, we have represented the animation with arrows.

Remarks: Good visualization can help explaining information better than textual output alone. We note that for particular COPs, e.g. Traveling Salesman Problem (TSP), one could come up with a natural problem-specific visualizations, e.g. TSP tour visualization. However such visualizations may not show the features of the fitness landscape or search trajectory well and furthermore, some problems may not have a natural problem-specific visualization.

Viz Experiment Wizard (EW)

To assist the user in preparing the SLS experiment design, executing the SLS runs, showing gross information about the runs, managing log files, and most importantly: to compute FLST visualization information as described above, we developed a tool called Viz EW, as seen in Fig 5.

Viz Single Instance Multiple Runs Analyzer (SIMRA)

To visualize the fitness landscape of a COP instance and one (or more — currently only two) SLS trajectory on the same instance, we use Viz **S**ingle (COP) **I**nstance **M**ultiple (SLS) **R**uns **A**nalyzer (SIMRA). It uses the visualization information produced by Viz EW and animates it in real time like a VCR according to the search time recorded in the RunLog file. The visuals in Fig 6, label 'A', give linked animations of FLST visualization as described in Fig 1-4, objective value over time, FDC visualizations, algorithm specific visualizations (eg. tabu tenure over time), and problem specific visualizations (eg. QAP data matrices structure). It also displays textual information – see Fig 6, label 'B'.

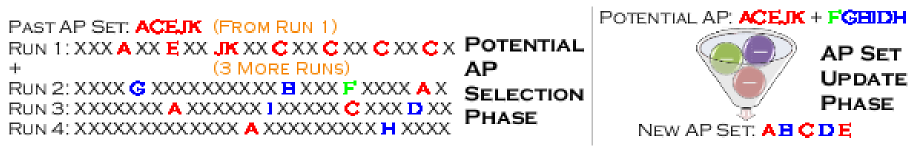

Past AP Set: ACEJK (From Run 1)
Run 1: XXX A XX E XX JK XX C XX C XX C XX C X
+ (3 More Runs)
Run 2: XXXX G XXXXXXXXXX B XXX F XXXX A X
Run 3: XXXXXXX A XXXXXX I XXXXX C XXX D XX
Run 4: XXXXXXXXXXXXX A XXXXXXXXX H XXXX
Potential AP Selection Phase
Potential AP: ACEJK + FGBIDH
AP Set Update Phase
New AP Set: AB C D E

Fig. 1. Potential AP Collection Phase and AP Set Update Phase. Let ACEJK be 5 APs collected from Run 1. Now, we execute 3 more SLS runs. The letters in red, green, blue indicate potential APs. After the AP Set Update Phase, 'B/D' replaces 'J/K'.

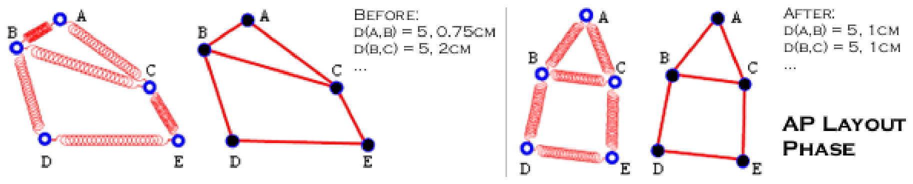


Fig. 2. AP Layout Phase using spring model. This follows gestalt law of proximity [20], APs that are near (far) with each other are viewed as clustered (spread).

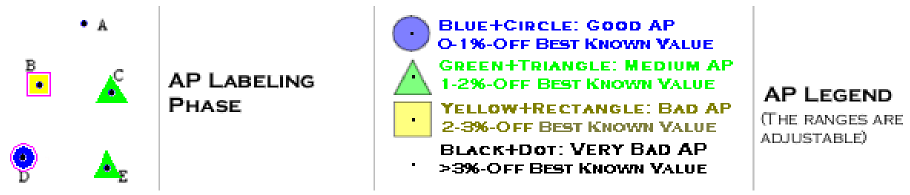

A
B
C
D
E
AP Labeling Phase
Blue+Circle: Good AP
0-1%-Off Best Known Value
Green+Triangle: Medium AP
1-2%-Off Best Known Value
Yellow+Rectangle: Bad AP
2-3%-Off Best Known Value
Black+Dot: Very Bad AP
>3%-Off Best Known Value
AP Legend
(The ranges are adjustable)

Fig. 3. AP Labeling Phase. This follows gestalt law of similarity [20]. APs with similar color and shape are grouped together by human perception system. The perception laws of proximity and similarity are utilized in this fitness landscape visualization.

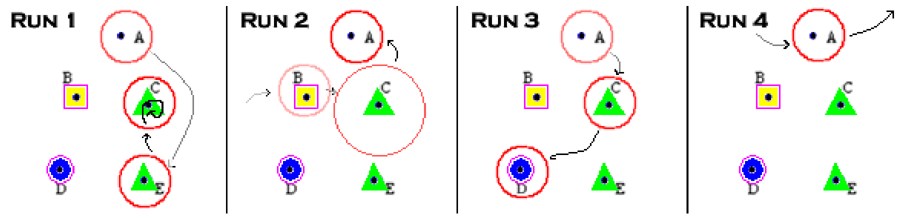
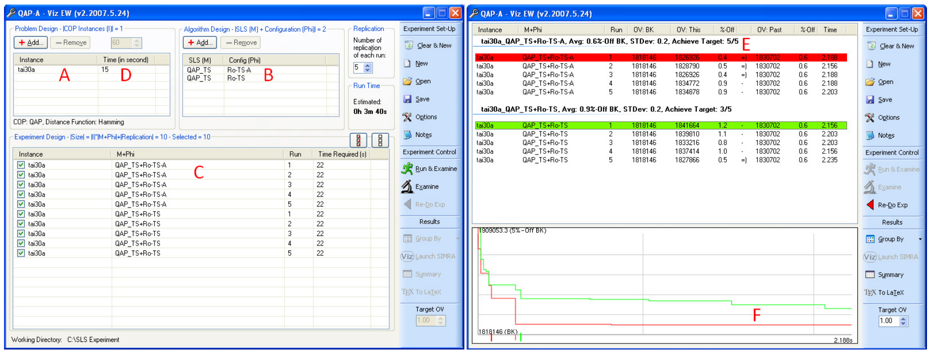


Fig. 4. Search Trajectory Layout Phase. The animation uses gestalt law of closure [20]. When the search trajectory is near an AP A and then quickly moves to AP B, human perception will fill in the details that the search traverses points along AP A→AP B. **Run 1**→SLS starts from a very bad AP A, walks to a medium quality AP E, and then cycles near AP C. **Run 2**→SLS starts from a bad AP B, then walks to a point near AP C (assume in Fig 1, the green point F in SLS run 2 happens to be near AP C), then moves to a very bad AP A (poor intensification). **Run 3**→SLS starts from a very bad AP A, but gradually walks to a medium AP C, escapes from it, then arrives at a good AP D (good intensification, compare this to Run 1 which is stuck in an attractive AP C). **Run 4**→The search trajectory of the SLS only bypasses a very bad AP A and is not near any other known APs (failure to navigate to promising region).

For other details about Viz GUI and visual features that are not included in this paper, please see `http://www.comp.nus.edu.sg/~stevenha/viz` or [5, 13–15].


QAP-A - Viz EW (v2.2007.5.24)

Problem Design - |COP Instances (I)| = 1

Instance	Time (in second)
tai30a	15

COP: QAP, Distance Function: Hamming

Algorithm Design - |SLS (M) + Configuration (Phi)| = 2

SLS (M)	Config (Phi)
QAP_TS	Ro-TS-A
QAP_TS	Ro-TS

Replication: Number of replication of each run: 5

Run Time: Estimated: 0h 3m 40s

Experiment Design - |Size| = |I|*|M+Phi|+|Replication| = 10 - Selected = 10

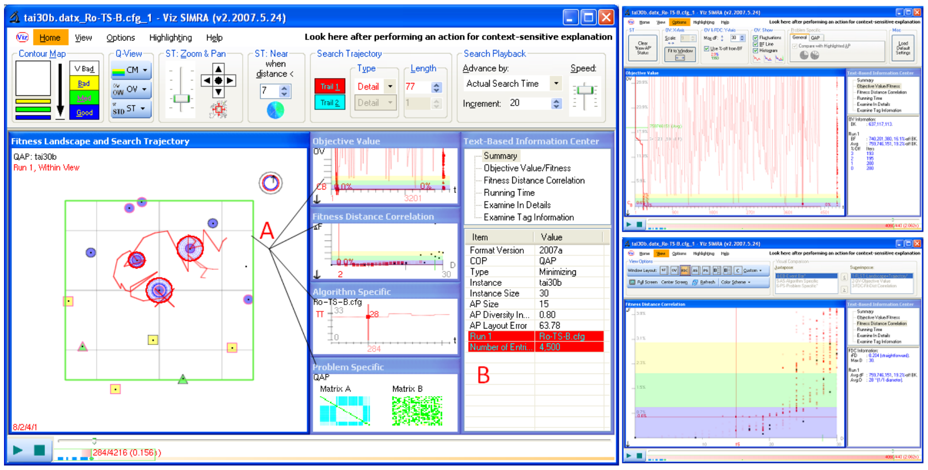
Instance	M+Phi	Run	Time Required (s)
✓ tai30a	QAP_TS+Ro-TS-A	1	22
✓ tai30a	QAP_TS+Ro-TS-A	2	22
✓ tai30a	QAP_TS+Ro-TS-A	3	22
✓ tai30a	QAP_TS+Ro-TS-A	4	22
✓ tai30a	QAP_TS+Ro-TS-A	5	22
✓ tai30a	QAP_TS+Ro-TS	1	22
✓ tai30a	QAP_TS+Ro-TS	2	22
✓ tai30a	QAP_TS+Ro-TS	3	22
✓ tai30a	QAP_TS+Ro-TS	4	22
✓ tai30a	QAP_TS+Ro-TS	5	22

Working Directory: C:\SLS Experiment

QAP-A - Viz EW (v2.2007.5.24)

Instance	M+Phi	Run	OV: BK	OV: This	%-Off		OV: Past	%-Off	Time
tai30a_QAP_TS+Ro-TS-A, Avg: 0.6%-Off BK, STDev: 0.2, Achieve Target: 5/5									
tai30a	QAP_TS+Ro-TS-A	1	1818146	1826926	0.4	=)	1830702	0.6	2.188
tai30a	QAP_TS+Ro-TS-A	2	1818146	1828790	0.5	=)	1830702	0.6	2.156
tai30a	QAP_TS+Ro-TS-A	3	1818146	1826926	0.4	=)	1830702	0.6	2.188
tai30a	QAP_TS+Ro-TS-A	4	1818146	1834772	0.9	-	1830702	0.6	2.188
tai30a	QAP_TS+Ro-TS-A	5	1818146	1834878	0.9	-	1830702	0.6	2.203
tai30a_QAP_TS+Ro-TS, Avg: 0.9%-Off BK, STDev: 0.2, Achieve Target: 3/5									
tai30a	QAP_TS+Ro-TS	1	1818146	1841664	1.2	-	1830702	0.6	2.156
tai30a	QAP_TS+Ro-TS	2	1818146	1839810	1.1	-	1830702	0.6	2.203
tai30a	QAP_TS+Ro-TS	3	1818146	1833216	0.8	-	1830702	0.6	2.203
tai30a	QAP_TS+Ro-TS	4	1818146	1837414	1.0	-	1830702	0.6	2.156
tai30a	QAP_TS+Ro-TS	5	1818146	1827866	0.5	=)	1830702	0.6	2.235

Fig. 5. Left: Viz EW has a GUI to specify: problem **(A)**, algorithm **(B)**, experiment **(C)** design, and time limit per instance **(D)**. **Right**: Results can be grouped with *context-sensitive* statistical data in the group heading **(E)**. The objective value summary visualization **(F)** gives a rough overview of the performance of the runs at a glance. The runs can be analyzed in detail with Viz SIMRA (see Fig 6).


tai30b.datx_Ro-TS-B.cfg_1 - Viz SIMRA (v2.2007.5.24)

Look here after performing an action for context-sensitive explanation

Search Playback: Advance by: Actual Search Time; Increment: 20

Item	Value
Format Version	2007a
COP	QAP
Type	Minimizing
Instance	tai30b
Instance Size	30
AP Size	15
AP Diversity In...	0.80
AP Layout Error	63.78
Run 1	Ro-TS-B.cfg
Number of Entri...	4,500

Fig. 6. The screen shot of Viz SIMRA for FLST and other visualizations.

3 A Step by Step Walk-Through with Ro-TS for QAP

In this section, we give an *in-depth* walk-through which shows how to incorporate the visualizations in Viz into the entire design and tuning of the SLS development process. Our objective is to show how visualization allows us to more easily understand the COP fitness landscape and the SLS trajectory. This then allows new improvements to the SLS based on the insights gained from visualization. We also show Viz features for simplifying the SLS development process.

The COP used is the Quadratic Assignment Problem (QAP) with benchmark instances from QAPLIB [21]. We have purposely chosen a classic COP for this walk-through so that the reader can appreciate the walk-through and the results easier. We know the best known (BK) objective values for each QAP instances in this benchmark library. From this, we have defined the following solution quality measures: good ($< 1\%$-off BK), medium ($1\% - 2\%$-off BK), bad ($2\% - 3\%$-off BK), and very bad ($> 3\%$-off BK) points.

For the experiments, we have fixed the number of iterations of every SLS run to be quite 'small': $5n^2$ iterations, where n is the instance size. This is because the state of the art SLS (including the one that we used) can obtain (near) optimal solutions for many QAPLIB instances with long runs. The goal of this experiment is to design and tune the best SLS+configuration for attacking the selected QAP instances within that *limited* $5n^2$ iteration bound.

We remark that for the purpose of simplifying the walk-through and reducing the number of visuals, we have taken the liberty of presenting this walk-through from the final step viewpoint in order to fit the walk-through within page constraints. Most of the FLST visualizations make use of the final AP set from good and bad runs from the entire development process. In the actual development process, we learn the fitness landscape structure and the search trajectory behavior incrementally. In sub-section 3.6, we give an example of our learning process for obtaining the final AP set used in the walk-through.

3.1. Experiment Set-Up: the QAP instances and baseline algorithm

We pick Taillard's QAP instances: tai30a/30b/35a/35b/50a/50b as training instances and tai40a/40b/60a/60b as test instances. These artificial QAP instances are chosen because the real life instances in QAPLIB are quite 'small' (≤ 36). The selected instances vary in size and have been generated using two strategies: uniformly generated matrices proposed in 1991 [17] and non-uniform matrices which resemble real-life instances proposed in 1995 [22]. For our purposes, we should imagine that initially we are unaware of the distinction between these two types, perhaps we are only at the 1991 time point.

There are several (successful) SLS proposals for QAP in the literature, for example: Robust Tabu Search (Ro-TS) [17]. We use Ro-TS as our baseline SLS — it is already a good SLS for QAP. Our implementation, which we called Ro-TS-I, uses a configuration similar to Ro-TS. The details are in Table 1.

We conduct some pilot runs. Ro-TS-I results are given in Table 2. It seems that within the limited iteration bound, the initial results are reasonably good for tai30a/35a/35b/50a but not for tai30b/50b (underlined). We want to investigate why this happens and find out how to adjust the SLS to get improvements.

3.2. Analyzing the fitness landscapes of QAP and Ro-TS-I behavior

Using the final AP set (see sub-section 3.6) and with a Hamming distance metric [18], we observe the fitness landscape of QAP instances as shown in Fig 7.

Component	**Choice**	**Remark**
Neighborhood	$O(n^2)$ 2-Opt	Natural (swap) move operation for QAP.
Objective Function	$O(1)$ delta	Measure delta as shown in [17].
Tabu Tenure (TT)	n	The default tabu tenure length.
Tabu 'Table' [17]	pair $i-j$	Item i cannot be swapped with j for TT steps.
Aspiration Criteria	Better	Override tabu if move leads to better solution.
Search Strategy	'Ro-TS' [17]	Change TT within [90%$*n$-110%$*n$] every $2n$ steps.

Table 1. Initial Ro-TS-I Configuration.

We observe a significant difference between tai30a (and tai35a/50a) and tai30b (and tai35b/50b). See figure text for details.

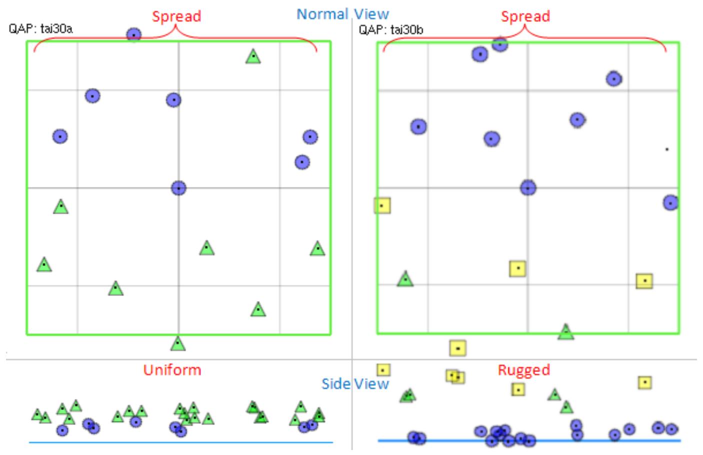


Fig. 7. Fitness landscape overview of two different types of QAP instances (tai30a and tai30b) with the *same* label setting (see Fig 3). The best found solution is always in the center. The anchor points in tai30a and tai30b are both spread throughout the fitness landscape and may be as far as the problem diameter (30 distance units). However, the quality of the anchor points in tai30b seems to be more 'rugged' than tai30a, especially when viewed using side view mode (the visualization is rotated 90 degrees along horizontal axis. The y-axis now shows the fitness of each AP w.r.t BK value).

A working hypothesis from this observation is that there are at least two classes of QAP instances. The next question is how to identify which instance belong to which class? Recall that we are not assuming that we know about the underlying problem instance generators. By looking at the fitness landscape visualizations and data matrices of the QAP instances, we classify tai30a/35a/50a (training), tai40a/60a (test) as QAP (type A) instances and tai30b/35b/50b (training), tai40b/60b (test) as QAP (type B) instances. This is consistent with the characteristics of these classes which differ in the smoothness (type A) or ruggedness (type B) in the fitness landscape visualization and in the uniformity (type A) and non-uniformity (type B) of their data matrices.

In Fig 9 (left) we observe that Ro-TS-I already has reasonable performance on the QAP (type A) instances. This may be because the gap among local optima is small — the quality of most APs are either blue circle (good) or green triangle (medium). The animations of search trajectories do not indicate any obvious sign of Ro-TS-I being stuck in a local optimum (explained textually: the search trajectory is found to be near one AP, quickly escapes from it, and later appears to be near another *different* AP). However, none of the Ro-TS-I runs gets the best known objective value within the limited iteration bound.

However, the performance of the same Ro-TS-I on the QAP (type B) instances is very bad. In Fig 10 (left) we observe that Ro-TS-I never visits any known good APs. Prior to obtaining the final AP set (see sub-section 3.6), we observe that Ro-TS-I often gets stuck in a bad local optimum region and cannot escape (explained textually: the search trajectory enters a region near one AP, then until the last iteration, it is still near the same AP). If that local optimum happens to be bad, the final best found solution reported will also be bad.

3.3. Hypotheses to improve walks on the QAP fitness landscapes

From the visualizations, the algorithm designers can arrive at the following two hypotheses on how the SLS should behave on these two classes:

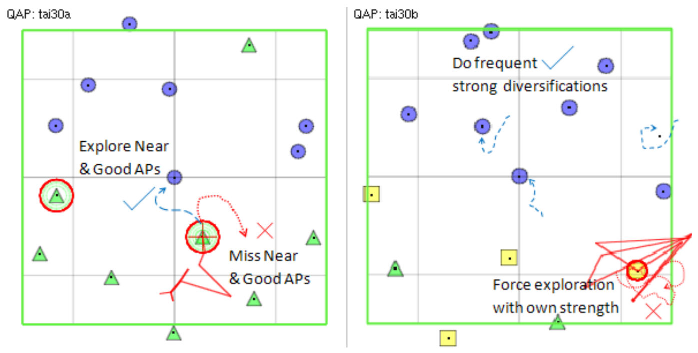


Fig. 8. Hypotheses of what the *idealized* good walks (blue dashed lines) should look like on QAP (type A/B) instances and the actual Ro-TS-I behavior (red dotted lines).

(1). QAP (type A) landscape is more smooth, so it is hard to decide where to navigate as 'everything' looks good. Diversifying too much may not be effective since we will likely end up in another region with similar quality. Our hypothesis: it is better for the SLS to reduce the possibility of missing the best solution within a close region where the SLS is currently in. Fig 8 (left) illustrates our hypothesis and shows the desired trajectory (blue dashed lines) searches around nearby good local optima rather than the trajectory of Ro-TS-I (red dotted lines) which moves away from the good local optima region.

(2). QAP (type B) landscape is more rugged, the local optima are deeper and spread out. Thus, we hypothesize that within the limited iteration bound, rather than 'struggling' to escape deep local optima with its own strength (e.g. via tabu mechanism), it is better for the SLS to do frequent strong diversifications as we know it is hard to escape from deep local optima and the 'nearest' local optima may be quite 'far' anyway. Fig 8 (right) illustrates our hypothesis where the desired trajectory (blue dashed lines) only makes short runs in a region before jumping elsewhere rather than the trajectory of Ro-TS-I (red dotted lines) which struggles to escape a deep local optimum.

3.4.a. Tweaking Ro-TS-I to Ro-TS-A for QAP (type A) instances

The search coverage (APs that are near any points in search trajectory are highlighted) of Ro-TS-I on QAP (type A) instances is not good (see Fig 9, left).

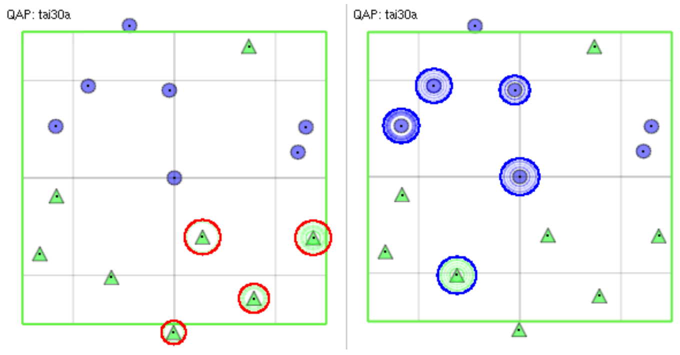


Fig. 9. Search coverage of Ro-TS-I (left) and Ro-TS-A (right) on QAP (type A) instance. Ro-TS-I (see large red circles) covers some of medium quality APs then wanders somewhere else far from known APs (animation not shown). On the other hand, lower Tabu Tenure Range in Ro-TS-A enables it to take different trajectory which covers some good quality APs (see large blue circles).

Visualization shows that Ro-TS-I actually gets near to known good APs but does not in the end navigate to those APs. Thus the good APs are missed. One reason may be that during short Ro-TS-I runs ($5n^2$ iterations), there are some Ro-TS-I moves leading to such good APs that are under tabu status and are not overridden by aspiration criteria.

The idea for robustness in Ro-TS [17] is in changing the tabu tenure randomly during the search within a defined Tabu Tenure Range (TTR). The TTR is defined by Tabu Tenure Low (TTL) and Tabu Tenure Delta (TTD) as follows: TTR=[TTL-(TTL+TTD)]. To encourage more intensification, we modify the Ro-TS-I by decreasing TTR from the recommendation in [17]: [90%*n-110%n] into a lower range and changing the robust tabu tenure value more often (after n steps, not after $2n$ steps, see Table 2).

Since we do not have information on the best TTR for Ro-TS-I, except that it should be lower, we experimented with various TTR settings to get TTL=40, TTD=40, TTR=[40%*n-80%*n] which performs well on the training instances and also slightly outperforms the original Ro-TS-I (see Table 2).

We call Ro-TS-I with lower TTR as Ro-TS-A. We observed that although TTR is smaller, it is still enough to ensure Ro-TS-A avoid solution cycling issues. This may be because it is quite easy to escape from any local optima of smooth fitness landscape of QAP (type A) instances. Fig 9 (right) shows the search coverage of Ro-TS-A which seems better than the search coverage of Ro-TS-I.

3.4.b. Tweaking Ro-TS-I to Ro-TS-B for QAP (type B) instances

Visualization reveals that Ro-TS-I does not manage to arrive near any of the known good APs. This leads to a very bad performance (see Fig 10, left).

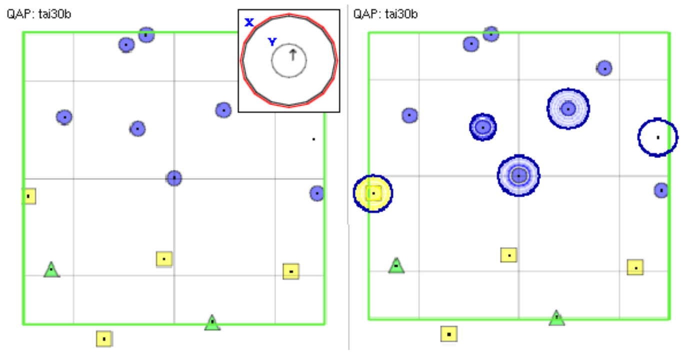


Fig. 10. Search coverage of Ro-TS-I (left) and Ro-TS-B (right) on QAP (type B) instance. Ro-TS-I (left) never arrives at any known good APs. On the other hand, Ro-TS-B (right) employs frequent strong diversifications and we see that it visits several APs of varying qualities (see large blue circles) that are (very) far from each other.

In Fig 10 (left), if circle ‘X’ is outside circle ‘Y’, it means that no AP is near the current point of the search trajectory of Ro-TS-I. The inability of Ro-TS-I to arrive at any known good APs combined with our understanding of QAP (type B) fitness landscape (sub-section 3.6) tells us that Ro-TS-I is stuck in a deep, bad local optima and fails to navigate to good region.

We have learned that in order to make Ro-TS-I behave more like our hypothesis, we need to add a strong diversification strategy. We tweak Ro-TS-I such that after n non improving moves, Ro-TS-I is considered to be stuck in a deep local optimum. To escape from it, we employ a *strong* diversification mechanism which preserves $max(0, n\text{-}X)$ items and randomly permutates the assignment of the other $min(n, X)$ items in the current solution. The value of X must be sufficiently large: close to n but not equal to n, otherwise it would be tantamount to random restart. The rationale for this *strong* diversification heuristic is that we see in the fitness landscape that blue circle (good) APs in type B instances are located quite far apart but *not* as far as the problem diameter n.

Now, it remains to determine the value of the diversification strength X? We experimented with different values of X on tai30b and obtained good results when X=20. However, for other training instance tai50b, the best value for X is when X=30. Visualization shows that even with X=20 (out of diameter n=50), Ro-TS-I still has problem in escaping local optima of tai50b.

From this, we realize that X should not be *fixed* for *all* QAP (type B) instances but rather be *robust* within a range correlated with the instance size. After further experimentation, we arrived at a good range for $X=[\frac{3}{4}n..\frac{7}{8}n]$ that performs well on the training instances. The value of X will be randomly changed within this range after each diversification step.

We call the revised SLS as Ro-TS-B. Animation shows that Ro-TS-B visits several APs with varying qualities, each is visited only in a brief period. Some APs visited by Ro-TS-B have good quality (see Fig 10 (right) and Table 2).

3.5. Benchmarking on the test instances

Finally, we verify the results against the test instances (also with the same iteration bound). The results are given in Table 2. On average, Ro-TS-A performs slightly better than Ro-TS-I on QAP (type A) instances while Ro-TS-B is significantly better than Ro-TS-I on QAP (type B) instances. The results here are also comparable to the updated Ro-TS results in [22].

We see that applying either Ro-TS-A or Ro-TS-B to opposite instance class has no or negative improvements (underlined). This shows that we have successfully tailored the SLS to match different fitness landscape of these instances.

Training Instances

Instance	n	Best Known	Iters	Time	Ro-TS-I		Ro-TS-A		Ro-TS-B	
					$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ
tai30a	30	1818146	4500	3s	1.1	0.4	**1.0**	0.3		
tai35a	35	2422002	6125	5s	1.1	0.2	**1.1**	0.1		
tai50a	50	4938796	12500	18s	1.5	0.1	**1.4**	0.3		
tai30b	30	637117113	4500	3s	<u>16.1</u>	0.0			**0.2**	0.1
tai35b	35	283315445	6125	5s	1.8	0.0			**0.3**	0.0
tai50b	50	458821517	12500	18s	<u>7.5</u>	0.0			**0.7**	0.2

Test Instances

Instance	n	Best Known	Iters	Time	Ro-TS-I $\bar{x}$	Ro-TS-I σ	Ro-TS-A $\bar{x}$	Ro-TS-A σ	Ro-TS-B $\bar{x}$	Ro-TS-B σ
tai40a	40	3139370	8000	8s	1.4	0.1	**1.0**	0.5	<u>2.0</u>	0.2
tai60a	60	7205962	18000	34s	1.7	0.1	**1.6**	0.2	<u>2.2</u>	0.3
tai40b	40	637250948	8000	8s	9.0	0.0	<u>9.0</u>	0.1	**0.0**	0.1
tai60b	60	608215054	18000	35s	2.1	0.1	<u>2.9</u>	0.2	**0.3**	0.1

Table 2. The results of Ro-TS-I, Ro-TS-A, and Ro-TS-B on training and test instances – averaged over **5** runs per instance. The instance size n, Best Known (BK) objective value, maximum iteration bound ($5n^2$), run times, average percentage-off $\bar{x}$ and standard deviation σ from BK are given.

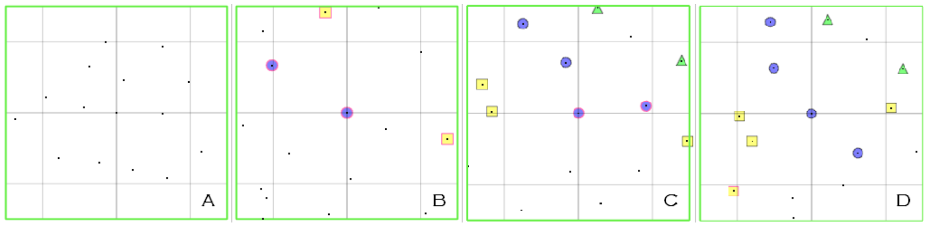


Fig. 11. The learning process for understanding QAP (type B) fitness landscape.

3.6. The learning process

The fitness landscape of a COP instance can be exponentially large. It is infeasible to enumerate all solutions just to analyze parts of the solutions visited by the SLS trajectory. As with other white-box approaches, FLST visualization must rely on progressive learning based on the solutions found by non-exact approaches. Here, we show our learning example for QAP (type B) instances:

Starting with Ro-TS-I, we obtained a fitness landscape (see Fig 11. **A**) with APs that are uniform, very bad (because we know the BK value), but not too far from each other (average pairwise distance between APs is $\approx \frac{1}{3}n$). We were curious why Ro-TS-I was stuck and added random restart diversification. Then, Ro-TS-I visited new (pink borders) and better APs (see Fig 11. **B**) which are quite far from the previous very bad APs. These better points are located in a deep valley as the fitness gap between them and their immediate neighbors are quite big. Soon, we learned that QAP (type B) has multiple deep valleys scattered far apart (see Fig 11. **C, D**). The distance between good APs are quite far but not $\approx$ diameter n. This tells us that good APs have similar substructures which inspires the strategy in Section 3.4.b.

For instances where BK values are not known (especially for new problems), we know less but we can still evaluate SLS performance w.r.t to what we have, e.g. in QAP (type B) example above, finding a new (better) AP that is far from any poorer APs that are known so far indicates the need of diversification.

4 Conclusion

In this paper, we have described Viz system and have shown an extended walk-through using Viz. The walk-through shows how to design and tune a baseline Ro-TS algorithm for the QAP. The insights from visualization results in two improved variants. We believe that the example is realistic and demonstrates why visualization is an interesting approach for designing and tuning SLS.

Viz is still under development but the prototype sytem can be downloaded from: `http://www.comp.nus.edu.sg/~stevenha/viz`.

Acknowledgements

We would like to thank the reviewers and the editors for their constructive inputs to improve the paper.

References

1. Charon, I., Hudry, O. Mixing Different Components of Metaheuristics. In: Meta-Heuristics: Theory and Applications. Kluwer (1996) 589–603
2. Birattari, M.: The Problem of Tuning Metaheuristics as seen from a machine learning perspective. PhD thesis, University Libre de Bruxelles (2004)
3. Hoos, H., Stuetzle, T.: Stochastic Local Search: Foundations and Applications. Morgan Kaufmann (2005)
4. Adenso-Diaz, B., Laguna, M.: Fine-tuning of Algorithms Using Fractional Experimental Designs and Local Search. Operations Research **54(1)** (2006) 99–114
5. Halim, S., Lau, H. Tuning Tabu Search Strategies via Visual Diagnosis. In: Meta-Heuristics: Progress as Complex Systems Optimization. Kluwer (2007)
6. Monett-Diaz, D.: +CARPS: Configuration of Metaheuristics Based on Cooperative Agents. In: International Workshop on Hybrid Metaheuristics. (2004) 115–125
7. Hutter, H., Hamadi, Y., Hoos, H., Leyton-Brown, K.: Performance Prediction and Automated Tuning of Randomized and Parametic Algorithms. In: International Conference on Principles and Practice of Constraint Programming. (2006) 213–228
8. Merz, P.: Memetic Algorithms for Combinatorial Optimization: Fitness Landscapes & Effective Search Strategies. PhD thesis, University of Siegen, Germany (2000)
9. Bartz-Beielstein, T.: Experimental Research in Evolutionary Computation: The New Experimentalism. Springer (2006)
10. Klau, G., Lesh, N., Marks, J., Mitzenmacher, M.: Human-Guided Tabu Search. In: National Conference on Artificial Intelligence (AAAI). (2002) 41–47
11. Syrjakow, M., Szczerbicka, H.: Java-based Animation of Probabilistic Search Algorithms. In: International Conference on Web-based Modeling and Simulation. (1999) 182–187
12. Kadluczka, M., Nelson, P., Tirpak, T.: N-to-2-Space Mapping for Visualization of Search Algorithm Performance. In: International Conference on Tools with Artificial Intelligence. (2004) 508–513
13. Lau, H., Wan, W., Halim, S.: Tuning Tabu Search Strategies via Visual Diagnosis. In: Metaheuristics International Conference. (2005) 630–636
14. Halim, S., Yap, R., Lau, H.: Visualization for Analyzing Trajectory-Based Metaheuristic Search Algorithms. In: European Conference on Artificial Intelligence. (2006) 703–704
15. Halim, S., Yap, R., Lau, H.: Viz: A Visual Analysis Suite for Explaining Local Search Behavior. In: User Interface Software and Technology. (2006) 57–66
16. Schneider, J., Kirkpatrick, S.: Stochastic Optimization. Springer (2006)
17. Taillard, E.: Robust Tabu Search for the Quadratic Assignment Problem. Parallel Computing **17** (1991) 443–455
18. Schiavinotto, T., Stuetzle, T.: A Review of Metrics on Permutations for Search Landscape Analysis. Computers and Operation Research **34(10)** (2007) 3143–3153
19. Graphviz: Graph Visualization Software. `www.graphviz.org`
20. Ware, C.: Information Visualization: Perception for Design. Second edn. Morgan Kaufmann (2004)
21. QAPLIB: Quadratic Assignment Problem Library. `www.seas.upenn.edu/qaplib`
22. Taillard, E.: Comparison of Iterative Searches for the Quadratic Assignment Problem. Location Science **3** (1995) 87–105