An Integrated White+Black Box Approach for Designing and Tuning Stochastic Local Search

Steven Halim[1], Roland H.C. Yap[1] and Hoong Chuin Lau[2]

[1] School of Computing, National University of Singapore
{stevenha,ryap}@comp.nus.edu.sg
[2] School of Information Systems, Singapore Management University
hclau@smu.edu.sg

Abstract. Stochastic Local Search (SLS) is a simple and effective paradigm for attacking a variety of Combinatorial (Optimization) Problems (COP). However, it is often non-trivial to get good results from an SLS; the designer of an SLS needs to undertake a laborious and ad-hoc algorithm tuning and re-design process for a particular COP. There are two general approaches. Black-box approach treats the SLS as a black-box in tuning the SLS parameters. White-box approach takes advantage of humans to observe the SLS in the tuning and SLS re-design. In this paper, we develop an integrated white+black box approach with extensive use of visualization (white-box) and factorial design (black-box) for tuning, and more importantly, for designing arbitrary SLS algorithms. Our integrated approach combines the strengths of white-box and black-box approaches and produces better results than either alone. We demonstrate an effective tool using the integrated white+black box approach to design and tune variants of Robust Tabu Search (Ro-TS) for Quadratic Assignment Problem (QAP).

1 Introduction

Stochastic Local Search (SLS) algorithms, also called Metaheuristics (e.g. Tabu Search, Iterated Local Search, etc) [1], have been extensively used to tackle large-scale NP-hard Combinatorial (Optimization) Problems (COP), often with impressive results. However, algorithm designers usually need to spend substantial effort to design and tune the SLS implementations to get good results.

Real world COPs are often new problems or variants of classic COPs but may not be well studied or no algorithm is known. It is frustrating that even if the COP C' resembles a *classic* COP C for which a good SLS S is known, a direct application of SLS S on COP C' will usually not immediately yield good performance. Thus, it is necessary to adapt an existing or create a new SLS for COP C'. 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 problem instances [1–5].

Most research has *only* focused on the problem of fine-tuning the parameter values for SLS. Pure tuning assumes that the appropriate SLS components and search strategies are already known and we just need to find the appropriate parameters for those components and strategies. In this paper, 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.

There are two general approaches for this SLS design and tuning problem [5]:

1. The *black-box approach* uses automatic fine-tuning. It aims to develop special 'tuning algorithms' to explore the SLS configuration space systematically and as efficient as possible. The human designs the SLS and specifies an initial configuration space to be explored by the automated tuning algorithm. The tuning algorithm finds the best configuration within the *given* configuration space and computational resource requirements. Examples of the black-box approaches are F-Race [3], CALIBRA [4], Search Parameter Optimization [6], and iMDF [7].
2. The *white-box approach* leverages the use of human intelligence and/or visual perception. It aims to create tools or methods to help analyze the performance of the SLS so that the algorithm designer has a basis for tweaking the SLS implementation. Thus, the SLS may be redesigned to extend beyond the initial configuration space. Examples of the white-box approaches are: Statistical Analysis (Fitness Distance Correlation, Run Time Distribution, etc) [1, 8], Sequential Parameter Optimization [9], Viz [10], etc.

Neither approach addresses *all* aspects of the SLS design and tuning. Black-box approaches are simple to apply, but will not help if the best configuration happens to be 'outside the box' of the initial configuration space (e.g. instance A with size 30 requires configuration 1, instance B with size 50 requires configuration 2 but the best configuration is a function of the instance size rather choosing between configuration 1 or 2). White-box approaches can give the algorithm designer insights into the search process, but are less effective for fine-tuning.

In this paper, we propose an integrated white+black box approach to address the SLS design and tuning problem. The fitness landscape and search trajectory visualization [10, 11] opens the SLS 'box' to allow better understanding of the current 'problem(s)' being faced by the SLS trajectory when searching in the fitness landscape of the COP instance. This allows the algorithm designer to narrow down the potentially huge set of SLS configuration space into a much smaller and focused configuration space. Black-box tuning then further fine-tunes the SLS on this focused configuration space using factorial design [12]. This process is iterated: use the visualization to verify the *new* SLS behavior, obtain more complete picture of the fitness landscape, generate new hypothesis or redesign the SLS if necessary, fine-tune the SLS with black-box tuning, and so on until the performance criteria are met.

Ideally our approach is valuable when facing new or not well understood problems. However, it would be more difficult to evaluate our approach on such problems since not enough may be known. As such, we evaluate our approach on classic COPs: the Traveling Salesman Problem (TSP) for illustration and the Quadratic Assignment Problem (QAP). We demonstrate that one can combine the strengths of both white+black box approaches to get better results than either alone. Furthermore, we take into account the practical effort and resource requirements for industrial problems: we want to design and tune SLS for good results within *limited development time* and *limited running time*.

We have developed a tool Viz for our integrated white+black box approach. Viz can visualize the COP fitness landscape and the SLS trajectory on the fitness landscape. Black-box tuning is supported using factorial design on the given configuration space. The details of the tools are outside the scope of this paper, see [10, 11]. The website `http://sls.visualization.googlepages.com` contains further details and also a video which complements the presentation here. The online PDF version of this paper is in color and can be magnified.

We remark that systematic search algorithms usually also employ heuristics. As an example, MULTI-TAC [13] configures search heuristics for backtracking search by learning configuration rules which are applied on backtracking search algorithm schemas. While our approach is not directly relevant to configure exact algorithms, combining human intelligence and searching in the algorithm configuration space is a promising avenue for further research.

2 The Basic Ideas

2.1. White-box: Fitness Landscape and Search Trajectory Analysis

The notion of fitness landscape has been shown to be useful for understanding the behavior (search trajectory) of SLS algorithms [1, 8, 14].

Given a COP instance π, the *fitness landscape* of π, $FL(\pi)$, is defined as the tuple $\langle S(\pi), d(s_1, s_2), g(\pi)\rangle$ [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. bond distance for TSP, Hamming distance for QAP, etc); and $g(\pi) : S \rightarrow \Re$ is the objective function. One can think of the fitness landscape as a surface with solutions as points on the surface, points are separated according to their distance, and the height 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 $(s_0, s_1, \ldots, s_k)$ where $s_i \in S(\pi)$ and $\forall i \in \{1, 2, \ldots, k\}$, (s_{i-1}, s_i) is either a local move done by the SLS according to its neighborhood $N(\pi)$ or a stronger diversification move beyond $N(\pi)$ [1]. Note that in this definition, a solution s_i can be satisfiable or not.

Understanding the characteristics of the fitness landscape empowers the algorithm designer to tailor the SLS implementation so that the search performs a better trajectory on the fitness landscape [1, 8, 14]. However, matching an SLS algorithm to the fitness landscape(s) of an instance or class of instances of a COP is not easy:

1. Different problem instances of the same COP may have quite different fitness landscapes [1].
2. The SLS behavior depends on the fitness landscape [1, 8, 14]. SLS with the same configuration may behave differently on different fitness landscapes.

3. The selected configuration (parameters [3, 4], heuristics components [2], and search strategies [5]), the implementation details, and any unexpected programming bugs, determine the *actual* SLS behavior. This behavior may be wrong, e.g. doing diversification when we expect the SLS to do intensification on the fitness landscape — the metaheuristic failure modes [15]).
4. Stochastic elements mean the SLS can take different search trajectories in replicated runs.

White-box techniques such as Fitness Distance Correlation (FDC) [1, 8] for analyzing fitness landscape properties and Run Time Distribution (RTD) [1, 16] for analyzing potential search stagnation, are commonly used to assist matching an SLS with the fitness landscape. However, while FDC and RTD are useful, they do not explain the *details* of the SLS behavior on the COP fitness landscape.

In [10, 11], we show how the abstract fitness landscape and search trajectory visualization in Viz can enhance the understanding of the SLS behavior on the COP fitness landscape. The main visualization ideas embedded inside Viz are briefly explained below.

Viz uses a notion of *Anchor Point* (AP) set which is a fixed set of local optima found by SLS runs on a COP instance. The AP set are diverse, high quality, and important solutions in the search runs (e.g. best found, frequently visited). The visualization layouts the AP points in an abstract 2-D space according to their distance metric w.r.t other APs. This forms the landscape visualization. Next, for each SLS run, we plot the positions of every solution or point in the search trajectory w.r.t its distance to AP points in the fitness landscape visualization. However, points that are too far from known APs are not visualized. See Fig 1 for the illustration of this visualization.

Our integrated white+black box approach uses Viz to first *understand* the characteristics of the fitness landscape of the COP (e.g. the fitness landscape is rugged). This helps the algorithm designer in *predicting* which search strategies will likely work well on the fitness landscape of the COP instance. The prediction can then be *verified* via visualization to see whether the SLS encounters any 'problem(s)' (e.g. stuck in a local optima as in Fig 1, Run 1). The algorithm designer uses these insights to think outside the box, make informed changes (e.g. adding a strong diversification strategy), and also narrow down the possible configuration space (e.g. avoid SLS configurations that make the SLS harder to escape from local optima such as lowering the tabu tenure in Tabu Search).

The white-box component leverages on the strengths of humans to analyze and learn from the visualizations. Although this is subjective and visualization is limited to points that have been visited, the process can be made intuitive and fruitful insights can be gained.

2.2. Black-box Tuning: Factorial Design

Ideally, given an initial SLS configuration space, black-box tuning algorithms can be used to systematically find the most suitable configuration in the given configuration space to attack the COP at hand. However in practice, the size of the SLS configuration space size may be huge. As such, the algorithm designer

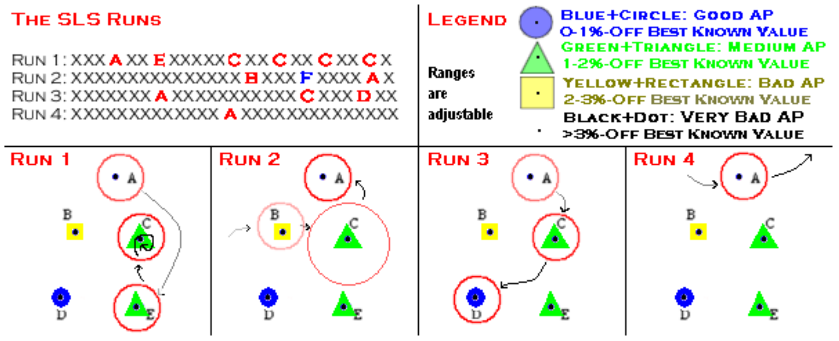
The SLS Runs
Run 1: XXX A XX E XXXXX C XX C XX C XX C X
Run 2: XXXXXXXXXXXXXXX B XXX F XXXX A X
Run 3: XXXXXXX A XXXXXXXXXXX C XXX D XX
Run 4: XXXXXXXXXXXXXX A XXXXXXXXXXXXXX
Legend
Ranges are adjustable
Blue+Circle: Good AP
0-1%-Off Best Known Value
Green+Triangle: Medium AP
1-2%-Off Best Known Value
Yellow+Rectangle: Bad AP
2-3%-Off Best Known Value
Black+Dot: Very Bad AP
>3%-Off Best Known Value
Run 1
Run 2
Run 3
Run 4
A
B
C
D
E

Fig. 1. An example of abstract fitness landscape and search trajectory visualization in Viz. The fitness landscape is represented by 5 APs {A,B,C,D,E} with quality labels shown in sub-figure 'Legend'. There are 4 SLS runs. **Run 1**→SLS starts from a very bad AP A, walks to a medium quality AP E, and then cycles around AP C. **Run 2**→SLS starts from a bad AP B, walks to a point *near* AP C (larger circle: assume blue point F in Run 2 is near AP C), then moves to a very bad AP A – a poor intensification. **Run 3**→SLS starts from a very bad AP A, gradually walks to a medium AP C, escapes from it, then arrives at a good AP D – a good intensification, better than Run 1 in escaping AP C. **Run 4**→The SLS trajectory bypasses a very bad AP A and is not near any other known APs – failure to navigate to promising region.

must give a 'sufficiently narrow' configuration space for the black-box tuning algorithm to work with since tuning time would otherwise take too long. Furthermore, if the best configuration happens to be 'outside the box' (the initial configuration space), then it cannot be found by fine-tuning alone.

Our integrated white+black box approach combines the strengths of both approaches to complement their weaknesses. After gaining insights into the fitness landscape and the SLS behaviors on the fitness landscape via visualization, the algorithm designers can use the insights to tweak the design of SLS, either by using *known* or *new* heuristic tweaks (e.g. adding a strong diversification strategy). This narrows down the configuration space substantially. The new algorithm can then be more easily fine-tuned (e.g. precisely how much diversification).

While a white-box approach may be usable for designing good SLS algorithm, it is still tedious for humans to explore the narrowed configuration space manually. Automatic black-box tuning algorithms are best for this situation. We have chosen to implement a full factorial design [12] in Viz system since we can obtain smaller configuration spaces through the white-box visualization process. However, other black-box tuning tools such as F-Race [3], CALIBRA [4], or iMDF [7] can be used in this phase.

3 An Integrated White+Black Approach

The methodology of the integrated white+black box approach is summarized in Fig 2. While the general approach is not new (compared with [9]), what is novel

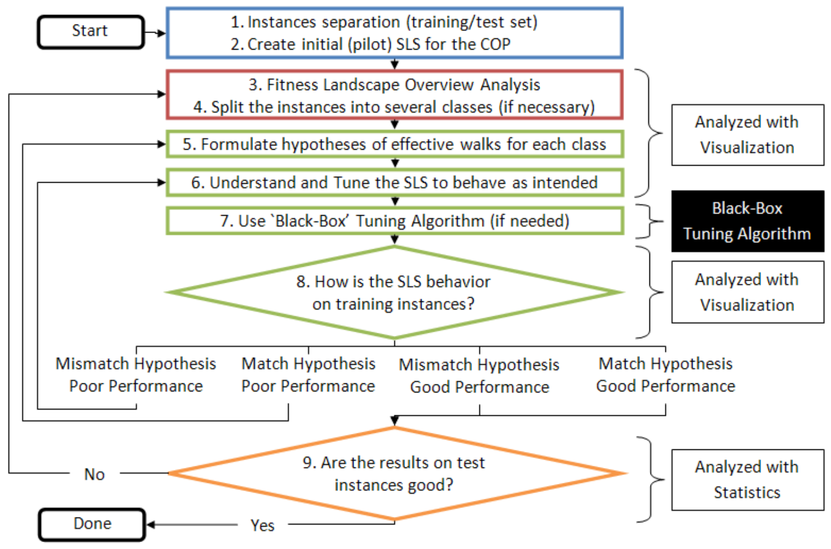
Start
1. Instances separation (training/test set)
2. Create initial (pilot) SLS for the COP
3. Fitness Landscape Overview Analysis
4. Split the instances into several classes (if necessary)
5. Formulate hypotheses of effective walks for each class
6. Understand and Tune the SLS to behave as intended
7. Use `Black-Box' Tuning Algorithm (if needed)
8. How is the SLS behavior on training instances?
Analyzed with Visualization
Black-Box Tuning Algorithm
Analyzed with Visualization
Mismatch Hypothesis Poor Performance
Match Hypothesis Poor Performance
Mismatch Hypothesis Good Performance
Match Hypothesis Good Performance
No
9. Are the results on test instances good?
Analyzed with Statistics
Done
Yes

Fig. 2. Flow chart of the integrated white+black box approach.

here is how visualization has been integrated in the white-box steps 3–6, 8, and black-box tuning algorithm in the black-box step 7. As we will show, the use of automatic visualizations in Viz[3] makes it much easier to analyze the fitness landscapes and SLS behaviors. Viz also has integrated support for black-box tuning in step 7 and some other handy automation for running SLS experiments and computing statistical analysis in step 9.

We first *briefly* illustrate some aspects of our integrated approach on the Traveling Salesman Problem (TSP). The chosen SLS is Iterated Local Search (TSP-ILS). It performs a 4-Opt perturbation, then the move operator swaps several pair of tour edges to reach a 2-Opt TSP local optimum. If the new local optimum is better, TSP-ILS will move to the new local optimum [16].

In Fig 3, we see that good quality (blue circle) and medium quality (green triangle) APs form one big cluster in the middle of the visualization (shown by the solid black arrows) and are close to each other when compared with the diameter of the fitness landscape (partially shown by the dashed red arrows). This shows a well known phenomenon called 'Big Valley'. We observe that outside this Big Valley region, we mostly see very bad (tiny black dot) APs. For such fitness landscape, it is suggested that the SLS should simply concentrate on the Big Valley region rather than wandering too far from it [1, 8, 16].

TSP-ILS already uses this strategy. However, visualization shows that sometimes it is stuck in a local optimum and unable to escape. Animation reveals that TSP-ILS is stuck in a place shown in Fig 3, label 'A'. This phenomenon is

[3] Viz includes visualizations such as the fitness landscape and search trajectory (see Fig 1), objective value over time, FDC scatter plot visualization, etc.

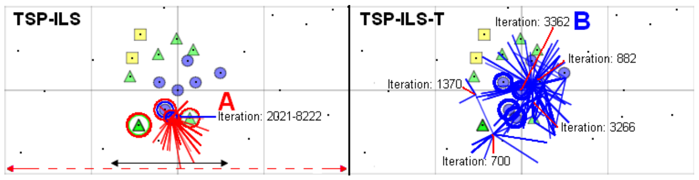

Fig. 3. Visualization of TSP fitness landscape and ILS behavior. See text for details.

also observable with RTD analysis by [16]. In [16], the authors suggested to use a stronger diversification than 4-Opt: 'FDD-diversification' after a cut-off time has elapsed without any improvement. Without going into details, the improved behavior is observable in Fig 3, label 'B' where the tweaked TSP-ILS-T is now able to escape from several local optima attractors and progresses closer towards the center of the screen (the best-known solution). With white-box analysis, one can derive reasonable variants of FDD-diversification. The range of the cut-off time can be predicted using white-box approaches like the RTD or visualization analysis above but the exact value is best determined with black-box tuning.

4 An Extended Case Study with Ro-TS for QAP

We use an extended case study where we apply our approach in Fig 2 to a realistic problem. It explains in more detail the individual steps in our approach.

We remark that in order to fit within the page constraints, we have taken the liberty of presenting this case study from the final step viewpoint. Most of the 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 via some pilot runs. For a discussion of incremental learning of fitness landscape and search trajectory with visualization, see [11].

4.1 Experiment Set-Up: QAP instances and baseline algorithm

The COP used is the Quadratic Assignment Problem (QAP) with benchmark instances from QAPLIB [17]. We have picked tai30a/30b/35a/35b/50a/50b as training instances and tai40a/40b/60a/60b/sko42/ste36b as test instances.

We have intentionally chosen a classic COP for this experiment so that the reader can more easily appreciate the problem and results. The best known (BK) objective values for each QAP instance are in the benchmark library. 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).

The initial baseline SLS for QAP in this case study is Robust Tabu Search (Ro-TS) [18] which has been shown to give good performance on QAP. We implemented a *variant* of Ro-TS called **Ro-TS-I** (Initial) which replicates the neighborhood and tabu mechanism in [18]. The details are in Table 1.

We do not expect to outperform the state-of-the-art algorithms on well studied and classic problems as they have been reached after extensive development

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 [18]
Tabu Tenure (TT)	n	The default tabu tenure length
Tabu 'Table' [18]	pair $i - j$	Item i cannot be swapped with item j for TT steps
Aspiration Criteria	Better	Override tabu if move leads to a better solution
Search Strategy	'Ro-TS' [18]	Change TT within [90%$*n$, 110%$*n$] after $2n$ steps

Table 1. Initial Ro-TS-I Configuration.

efforts over a long period of time. For QAP, many good SLS algorithms, including Ro-TS, can find the BK objective value of many QAP instances in QAPLIB with long runs. Thus, to prevent the *ceiling effect*, we have fixed the number of iterations of every SLS run to be quite 'small': $5n^2$ iterations where n is the instance size. The experiment goal is to design and tune the best SLS+configuration for attacking the selected QAP instances within that *limited* $5n^2$ iteration bound.

4.2. Preliminary Analysis

The initial results of Ro-TS-I on training instances are shown in Table 2. We observe 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 we get these results and redesign an improved SLS.

We apply fitness landscape visualization on the QAP training instances. We observe a possible difference between tai35a (and tai30a/50a) with tai35b (and tai30b/50b). See Fig 4 text for details.

A working hypothesis from this observation is that there are *at least* **two classes of QAP instances**. By looking at the fitness landscape visualizations and/or data matrices of the QAP instances, we classify tai30a/35a/50a (training), tai40a/60a/sko42 (test) as QAP (type A) instances and tai30b/35b/50b (training), tai40b/60b/ste36b (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 Table 2, we observe that Ro-TS-I already has reasonable performance on the type A instances. This may be because the gap among local optima is small — the quality of most APs are medium (green triangle). The animations of search trajectories of Ro-TS-I do not indicate any obvious sign of Ro-TS-I being stuck in a local optimum.

Since the QAP (type A) landscape is smoother, it is hard to decide where to navigate as 'everything' looks good. Diversifying too much may not be effective since the search will likely end up in another region with similar quality. We thus formulate the hypothesis that it is better to reduce the possibility of missing the best solution within a close region where the SLS is currently in. Fig 4 (left) illustrates our hypothesis and shows the ideal desired trajectory (blue dashed

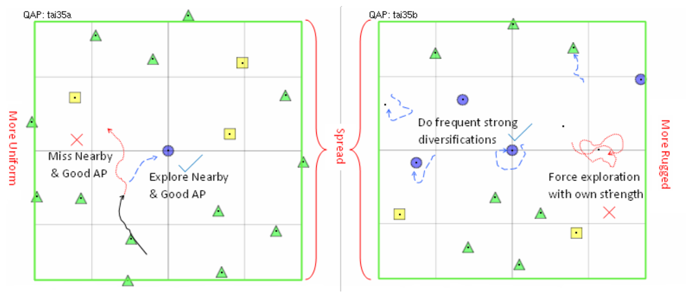

Fig. 4. Fitness landscape overview of tai35a and tai35b automatically generated from Viz. See Fig 1 sub-figure 'Legend' for the meaning of the shapes and colors. The best found solution is always in the center. APs in tai35a and tai35b are spread throughout the fitness landscape. However, the quality of the APs in tai35a seems to be more 'uniform' (most are green triangles) than tai35b (all types of AP quality exist). The actual Ro-TS-I behaviors on QAP (type A/B) instances are shown with red dotted lines. Our hypotheses on *ideal* good trajectories are shown with blue dashed lines.

lines) searches around nearby good local optima rather than Ro-TS-I trajectory (red dotted lines) which moves away from the good local optima region.

On the other hand, the performance of the same Ro-TS-I on the type B instances is very bad as seen in Table 2. In Fig 6 (left) we observe that Ro-TS-I is stuck in very bad APs (textual explanation of the visualization: the search trajectory enters a region near some APs, then until the last iteration, it is still near the same APs). If the quality of the solutions in that region happens to be bad, the final best found solution reported will also be bad.

The QAP (type B) landscape is more rugged, i.e. the local optima are deeper and more spread out. We thus hypothesize that within the limited iteration bound, rather than attempting to escape deep local optima with its own strength (e.g. via a tabu mechanism), it is better for the SLS to perform frequent strong diversification. Fig 4 (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.

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

The search coverage[4] of Ro-TS-I on QAP (type A) instances is not good (see Fig 5, left). Visualization[5] shows that Ro-TS-I sometimes gets near to known good APs but does not in the end navigate to those APs.

Initially, we thought that in order to make Ro-TS-I focus on a particular region, we should increase the intensification from a 2-Opt into a **3-Opt** swap move neighborhood. Although this is more costly, it might allow better results

[4] APs that are near any points in search trajectory are highlighted.

[5] In Viz, a circle drawn on an AP shows that the SLS trajectory pass through an area near that AP. The diameter of the enclosing circle shows the approximate distance.

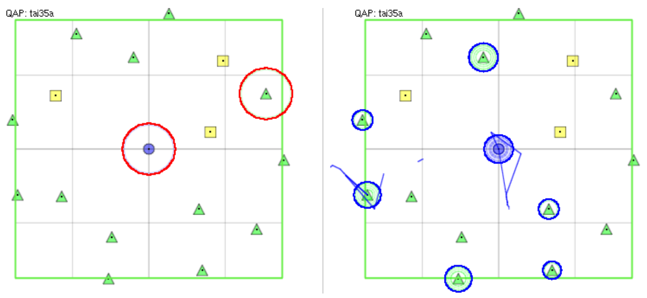

Fig. 5. Search coverage of Ro-TS-I (left) and Ro-TS-A (right) on QAP (type A) instance. Ro-TS-I (left) is already 'near' the best found AP but then wanders somewhere else (see the large red circles, animation not shown). On the other hand, a lower Tabu Tenure Range in Ro-TS-A (right) enables it to take different search trajectory which covers some medium and good quality APs (see the large blue circles).

from the region where the SLS is currently searching in. However, this idea turned out to be ineffective as no significant improvement in the search behavior was seen. To investigate, we added an algorithm specific visualization that shows a spike if 3-Opt code is executed. We were surprised that we almost never saw any spikes (only rarely when attacking tai35a). This may be because the smooth fitness landscape causes most moves that exchange 3 facilities at once in a QAP solution are worse than those that exchange 2 facilities at once. Also, the larger neighborhood slows down the SLS significantly. The results do not show significant improvement (see Table 2) so this configuration was not pursued.

We came up with another hypothesis for the SLS algorithm. During short runs, there may be some Ro-TS-I moves which lead to known good APs that are under tabu status and are not overridden by aspiration criteria.

The idea for robustness in Ro-TS [18] is to change tabu tenure randomly during the search within a defined Tabu Tenure Range (TTR) every Z*n steps. The TTR is defined as the interval $[TTL, TTL{+}TTD]$ which has two parameters: Tabu Tenure Low (TTL) and Tabu Tenure Delta (TTD). To encourage Ro-TS-I to do more intensification, we decrease its TTR from the recommendation in [18]: [90%*n, 110%n] into a lower range and changing the robust tabu tenure value more often — after n steps (Z=1), not $2n$ steps (Z=2) as in Table 1.

We do not know the best TTR for Ro-TS-I, except that it should be lower. We use systematic black-box tuning with *full factorial design* on TTL={40, 70} and TTD={20, 40} and obtain TTR=[40%*n, 80%*n] (TTL=40, TTD=40) as the TTR that works best on the training instances and also slightly but consistently 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 avoids solution cycling issues. This may be because it is quite easy to escape from any local optima of smooth fitness landscape of type A instances. Fig 5 (right) shows the search coverage of Ro-TS-A which seems better than the search coverage of Ro-TS-I.

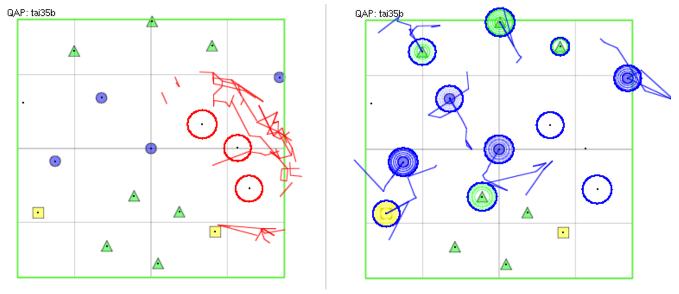

Fig. 6. Search coverage of Ro-TS-I (left) and Ro-TS-B (right) on QAP (type B) instance. Ro-TS-I (left) is stuck around very bad APs (see the large red circles). On the other hand, Ro-TS-B (right) employs frequent strong diversifications and we see that it visits several APs of varying qualities (see the large blue circles) that are (very) far from each other.

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

Visualization reveals that Ro-TS-I is stuck near very bad APs. This leads to a very bad performance (see Fig 6, left). With the understanding that the fitness landscape of type B instances is rugged, we conclude that the inability of Ro-TS-I to escape those APs is because the APs are part of deep local optima regions.

To alleviate this situation, we add a strong diversification strategy into the Ro-TS-I. We consider Ro-TS-I to be stuck in a deep local optimum after n non-improving moves (Z=1). To escape, 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 is 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 good APs (blue circles) in type B instances are located quite far apart but *not* as far as the problem diameter n.

How should the diversification strength X be determined? One can manually experiment with different values of X on various training instances but it is better to do systematic black-box tuning. We automatically tune using 1-factor design on X={5, 10, ..., n}. With tai30b and tai35b as training instances, we get good results when X=15. However, X=15 is not the best configuration for tai50b (see Table 2). We found that fixing the value of X to a constant (**Fixed-Diversification**) tends to make the Ro-TS-I overfit the training instances.

After fine-tuning the fixed-diversification strategy, we realized that X should not be *fixed* for all type B instances but rather be *robust* within a range correlated with the instance size. The value of X is randomly changed within this range after each diversification step. This helps maintaining the consistency of the performance quality across various QAP (type B) instances.

As the pilot runs using a fixed-diversification strategy yield reasonably good results when X is set around the half of the instance size n, we apply *full factorial design* on X_{low}=$\{\frac{4}{10}n, \frac{5}{10}n\}$ and X_{high}=$\{\frac{6}{10}n, \frac{7}{10}n\}$ to try various X within the

Training Instances

Instance	n	Best Known	Iters	Time	Ro-TS-I		> 3-Opt <		Ro-TS-A	
					$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ
tai30a	30	1818146	4500	3s	0.99	0.46	1.00	0.38	**0.85**	0.32
tai35a	35	2422002	6125	5s	1.24	0.22	1.14	0.16	**0.98**	0.29
tai50a	50	4938796	12500	18s	1.62	0.11	1.67	0.15	**1.42**	0.21
Instance	**n**	**Best Known**	**Iters**	**Time**	**Ro-TS-I**		**> X=15 <**		**Ro-TS-B**	
					$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ
tai30b	30	637117113	4500	3s	<u>16.18</u>	0.00	0.14	0.11	**0.17**	0.17
tai35b	35	283315445	6125	5s	2.90	1.06	0.20	0.14	**0.25**	0.26
tai50b	50	458821517	12500	18s	<u>7.58</u>	0.01	0.79	0.84	**0.15**	0.14

Test 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
tai40a	40	3139370	8000	8s	1.25	0.19	**1.22**	0.25	<u>1.63</u>	0.26
sko42	42	15812	8820	9s	0.21	0.08	**0.11**	0.06	0.16	0.09
tai60a	60	7205962	18000	34s	1.60	0.17	**1.53**	0.14	<u>2.12</u>	0.16
ste36b	36	15852	6480	6s	6.20	1.21	<u>7.28</u>	0.92	**0.65**	0.73
tai40b	40	637250948	8000	8s	9.01	0.00	<u>9.04</u>	0.09	**0.01**	0.02
tai60b	60	608215054	18000	35s	2.38	0.47	<u>2.93</u>	0.36	**0.17**	0.13

Table 2. The results of Ro-TS-I, Ro-TS-A, and Ro-TS-B on training and test instances — averaged over **10** runs per instance. The instance size n, Best Known (BK) objective value, maximum iteration bound ($5n^2$ iterations), run times (except column '3-Opt'), average percentage-off $\bar{x}$ and standard deviation σ from BK are given.

interval $[X_{low},\ X_{high}]$ settings systematically. We arrived at a good range for $X=[\frac{4}{10}n,\ \frac{6}{10}n]$ that works best on the training instances.

We call the revised SLS as **Ro-TS-B**. Animation shows that Ro-TS-B visits several far away APs of varied quality, where each AP is visited only in a brief period. However, some APs visited by Ro-TS-B have good quality and thus the overall performance is good. See Fig 6 (right) and Table 2.

4.5. Benchmarking on the test instances.

We now compare the initial and final SLS algorithms on the test instances using the same iteration bound. The results are given in Table 2. We observe that on average Ro-TS-A performs slightly better than Ro-TS-I on type A instances while Ro-TS-B is significantly better than Ro-TS-I on type B instances. Note that for type A instances, since the fitness landscape is more smooth, any improvements will be small. The results here are also comparable with the updated Ro-TS results in [19].

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

Sc	Training Set	Configuration	tai40a	sko42	tai60a	ste36b	tai40b	tai60b
1.	tai30**a**/tai30**b** tai50**a**/tai50**b**	{3, 2, 40, 10, 20}	1.23 (0.29)	**0.09** (0.08)	1.59 (0.20)	0.90 (0.80)	1.86 (1.55)	1.70 (0.24)
2.	tai30**b**/tai35**b** tai50**b**	{3, 1, 90, 40, -}	1.26 (0.16)	0.28 (0.09)	1.55 (0.21)	5.12 (1.30)	8.93 (0.16)	2.09 (0.02)
3.	tai30**a**/tai35**a** tai50**a**	{3, 1, 40, 40, -}	1.25 (0.19)	0.16 (0.06)	**1.50** (0.15)	7.13 (1.01)	9.04 (0.09)	2.93 (0.33)
4.	tai30**b**/tai35**b** tai50**b**	{1, 2, 90, 40, 20}	1.81 (0.24)	0.16 (0.11)	1.70 (0.16)	0.93 (0.89)	0.05 (0.09)	0.94 (0.71)

Table 3. CALIBRA results on tuning Ro-TS-I^C. The 1st column is the scenario ID. The 3rd column gives the selected configuration when CALIBRA is trained using training instances specified in the 2nd column. The 4th-9th columns give the percentage-off and standard deviation w.r.t BK values on the test instances — averaged over **10** runs.

5 Comparison with a Pure Black-Box Approach

We compared our integrated white+black approach with a pure black-box approach on several tuning scenarios. Note that we have not compared against a pure white-box approach as the comparison would be subjective.

We use CALIBRA [4] as the black-box tuning algorithm. CALIBRA works by iteratively trying different configurations from the given configuration space (using fractional factorial design) to set-up the SLS, run the SLS on training instances, and obtain results from the black-box SLS (e.g. best found solution quality). CALIBRA uses the results from the SLS runs to determine which configuration to try next.

For the experiments with CALIBRA, we chose the following *default* range of values for the initial configuration space {Z,Str,TTL,TTD,X} with size 1152 (3*2*8*6*4). The size of this configuration space is purposely larger and includes most of the configuration space used in Section 4. Note that this configuration would not be necessarily obvious from the initial Ro-TS-I configuration.

1. Execute strategy after **Z***n iterations without improvement, Z: {1, 2, 3},
2. Strategy (**Str**): {1: Lower-TTR, 2: Fixed-Diversification},
3. Tabu Tenure Low (**TTL**): {30, 40, . . . , 100},
4. Tabu Tenure Delta (**TTD**): {0, 10, . . . , 50},
5. Diversification strength **X** (only used when Str=2), X: {20, 25, . . . , 35}.

CALIBRA is used to configure our baseline algorithm Ro-TS-I for 20 minutes per scenario (that is, with $\approx$ 30 seconds for Ro-TS-I runs on QAP instances of size 30/35/50, this roughly allows CALIBRA to examine $\approx 20 * 60/30 \approx 40$ different configurations. This is sufficient for our experiments as CALIBRA seems to hit a local optimum of the configuration space after around 30 configurations. The various CALIBRA configured algorithms are called Ro-TS-I^C.

In scenario 1, we assume that we do not know that there are 2 different fitness landscape characteristics in QAP instances used. We see that CALIBRA

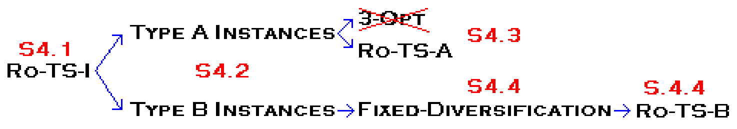

Fig. 7. The development of the two Ro-TS variants.

chooses a 'balanced' configuration {3, Fixed-Diversification, 40, 10, 20} when trained with mixed type A and type B instances. This Ro-TS-I^C yields *balanced* performance on both types but poorer than the specialized Ro-TS-A or Ro-TS-B on type A or type B instances, respectively (compare with Table 2).

In scenario 2, we deliberately omitted Fixed-Diversification as the choice for the second parameter (Str). By doing so, CALIBRA is forced to choose the 'best' configuration: {3, Lower-TTR, 90, 40, -} in the smaller box with size 576 when given type B training instances. This Ro-TS-I^C produces very bad results for type B test instances (compare with Ro-TS-B results in Table 2) and shows that pure black-box tuning algorithm cannot find a good configuration outside the box by itself.

Scenario 3 and scenario 4 are similar to the full factorial design using Viz in Section 4 but now with bigger configuration space. Note that CALIBRA uses fractional factorial design.

In scenario 3, since CALIBRA is trained with type A instances, it converges to a configuration that works well for type A but works badly for type B instances. Ro-TS-I^C (compare with Z=1 in the configuration of Ro-TS-A) obtains better results for tai60a but not for tai40a and sko42. However, the performance of Ro-TS-I^C is more or less similar to Ro-TS-A in this scenario.

The opposite situation from scenario 3 occurs in scenario 4 where this time the configured Ro-TS-I^C performs much better on type B instances. Note that since a black-box tuning algorithm cannot think out of the box to correlate X with instance size, it selects the 'best' X given the training instances. We get X=20 which is good for tai40b but bad for ste36b and tai60b — a case of over-fitting. Ro-TS-I^C has worse performance than Ro-TS-B in this scenario.

6 Conclusion

In this paper, we have shown an integrated white+black box approach for designing and tuning SLS algorithms. We demonstrate that starting from a good baseline SLS, Robust Tabu Search, on a realistic problem, QAP, we are able to derive two algorithm variants with better performance. Fig 7 summarizes the development steps with reference to SLS algorithms devised and tuned in Section 4. The white-box steps are necessarily subjective. However, since Robust Tabu Search is already quite good on QAP and given that each major step in the SLS algorithm development is well supported by visualizations and tools, we believe that this gives evidence that our approach is both valuable and effective.

In practice, for real-life COPs, the goal is to design an SLS algorithm with good performance under limited resources. To reduce the development time, we would also need techniques and more importantly tools which can help in SLS design and tuning. Our integrated white+black box approach combines the strengths of leveraging the developer intuition by using generic automated visualization tools with generic automatic parameter tuning. The techniques presented here should be relevant to anyone designing a new SLS, particularly when dealing with new COPs.

References

1. Hoos, H.H., Stuetzle, T.: Stochastic Local Search: Foundations and Applications. Morgan Kaufmann (2005)
2. Charon, I., Hudry, O. Mixing Different Components of Metaheuristics. In: Meta-Heuristics: Theory and Applications. Kluwer (1996) 589–603
3. Birattari, M.: The Problem of Tuning Metaheuristics as seen from a machine learning perspective. PhD thesis, Université Libre de Bruxelles (2004)
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.C. Tuning Tabu Search Strategies via Visual Diagnosis. In: Meta-Heuristics: Progress as Complex Systems Optimization. Kluwer (2007)
6. Hutter, F., Hamadi, Y., Hoos, H.H., Leyton-Brown, K.: Performance Prediction and Automated Tuning of Randomized and Parametic Algorithms. In: Principles and Practice of Constraint Programming. (2006) 213–228
7. Lau, H.C., Xiao, F.: Toward an Intelligent Metaheuristics Framework. In: Metaheuristics International Conference. (2007)
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. Halim, S., Yap, R.H.C., Lau, H.C.: Viz: A Visual Analysis Suite for Explaining Local Search Behavior. In: User Interface Software and Technology. (2006) 57–66
11. Halim, S., Yap, R.H.C.: Designing and Tuning SLS through Animation and Graphics - an extended walk-through. In: SLS: Stochastic Local Search Workshop. (2007)
12. NIST: e-Handbook of Statistical Methods. www.itl.nist.gov/div898/handbook
13. Minton, S.: Automatically Configuring Constraint Satisfaction Programs: A Case Study. Constraints **1**(1/2) (1996) 7–43
14. Schneider, J.J., Kirkpatrick, S.: Stochastic Optimization. Springer (2006)
15. Watson, J.P.: On Metaheuristics “Failure Modes”. In: Metaheuristics International Conference. (2005) 910–915
16. Stuetzle, T., Hoos, H.H. Analyzing the Run-Time Behavior of Iterated Local Search for the TSP. In: Essays and Surveys in Metaheuristics. Kluwer (1999) 449–453
17. QAPLIB: Quadratic Assignment Problem Library. www.seas.upenn.edu/qaplib
18. Taillard, E.: Robust Tabu Search for Quadratic Assignment Problem. Parallel Computing **17** (1991) 443–455
19. Taillard, E.: Comparison of Iterative Searches for the Quadratic Assignment Problem. Location Science **3** (1995) 87–105