Contents

Chapter 3

SLS Design and Tuning Problem

"Whoever loves money never has money enough;
Whoever loves wealth is never satisfied with his income."
— **Ecclesiastes 5:10a, Holy Bible [49]**

"He, who is greedy, is always in want."
— **Horace – Ancient Roman Poet**

"Aiming at the best is one of the most fundamental traits of intelligence...
The pursuit of the best appears so connatural with the human mind
that when we do not recognize it in somebody's behavior,
we readily qualify him or her as irrational."
— **[6], chapter 1, paragraph 1)**

In this chapter, we discuss the case of SLS Design and Tuning Problem *that will always be faced by the user of SLS algorithms. Parts of this chapter have been published in MIC05 [31, 17]*

3.1 The Quest for a Better Performance

The crux of the problem highlighted in this thesis is, not surprisingly, caused by human's inner desire as a creature of comparison: his greed for 'a better one'. And while greed is often considered as one of the deadly sins, it is, ironically, also the major source of motivation for many Computer Science researchers and practitioners on the quest for 'a better one'.

Simply said, the main problem discussed in this thesis: the SLS Design and Tuning Problem is merely another quest for a better (SLS) performance. The importance for a better SLS performance on various COPs is obvious, e.g. more efficiency, more time savings, more cost savings, and ultimately more profit.

While it is rather obvious to seek for a better performance if the current performance of the particular SLS algorithm is low on average or when it varies greatly across different test instances beyond acceptable level, it is also evident that people are still looking for a better performing SLS algorithm because they want to find better, or even the 'best'[1] SLS for their COPs, as witnessed in the literature about SLS algorithms.

If exact algorithms only allow room for improvement in terms of speed, SLS algorithms adds one more dimension in terms of improving solution quality because of their incompleteness — they do not have guarantee on optimality. Some SLS algorithms yield better solution quality in certain condition while others are better in another condition. Some run faster while some are easier to implement. Some are quite robust while some produce solutions quality with high degree of variability. There are various 'configure-able' parts inside heuristic-type algorithms. So, an obvious question arises: which one is the best in the current context? This issue is heightened if best published results are being sought for.

[1] A 'dream' heuristic-based algorithm that always produce the optimal solution in short time. This is an ideal combination of the power of the optimality of exact algorithms with the low cost and flexibility of non-exact heuristic algorithms.

Although the existence of the SLS Design and Tuning Problem and the crucial importance of good algorithm design and proper tuning are acknowledged in the literature since a long time ago, related works around SLS Design and Tuning Problem have only emerged *recently*, as shown in the time line of related works, for example:

- Racing algorithms - Birattari (2002-2004) [7, 6], Yuan and Gallagher (2005) [48]
- CALIBRA - Adenso-Diaz and Laguna (2002-2006) [1]
- Agent configuration algorithm - Monett-Diaz (2004) [36]
- ParamILS - Hutter *et al.* (2007) [24]
- Our works (2004-2008) [31, 17, 19, 20, 18, 21].

It seems that this quest for a better performance is attracting more and more participants. We believe that this research topic has been (and likely to remain true for few years ahead) a fertile ground for breeding new knowledge to advance the field of optimization algorithms.

3.2 Formal Definition and Classification

Formal Definition of SLS Design and Tuning Problem

Let:

M : The basic algorithmic template of an SLS

ϕ : The configuration of SLS: (parameter values, components, and search strategies). See Chapter 2.3.4 for more details about $M + \phi$.

I_{train} :[2] The known set of COP training instances faced by the SLS *now*.

I_{test} : The known set of COP test instances faced by the SLS *now*.

I_{future} : The unknown set of COP future instances that will be faced by the SLS *later*.

T_{dev} : The *tight* development time for designing, implementing, and tuning the SLS.

T_{run} : The *tight*[3] running or computation time for the SLS to attack the given set of training instances I_{train} or test instances I_{test}.

Then, SLS Design and Tuning Problem[4], abbreviated as **SLS DTP**, is defined as a multi-objectives multi-constraints problem:

> The problem of finding a good combination $M + \phi$, within the tight development time T_{dev}, in order to obtain an SLS algorithm $M + \phi$ that:
>
> 1. can obtain high quality (acceptable) solutions,
> 2. robust, and
> 3. has fast run time,
>
> when trained using I_{train} and tested using other set I_{test} of the COP being attacked within limited running time T_{run}. By the regularity of nature, the set I_{future} is *expected* to have similar characteristic with I_{train} and I_{test} and thus the performance of the SLS $M + \phi$ on I_{future} is *expected* to be more or less similar.

From the definition above, SLS DTP is actually another 'search problem', but in configuration space. Since there is no well defined rules yet for picking the best $M + \phi$, addressing SLS DTP is a process to learn the relationship of $M + \phi_1$, $M + \phi_2$, ..., or $M + \phi_n$ w.r.t the performance of the SLS algorithm while being used to attack various instances of the COP. Such relationships are then used as the basis for picking the most appropriate $M + \phi$ pair in the current context. It is not easy to address SLS DTP as the size of configuration space is big. Philosophically, we believe that

[2] Note that the selection of the training and test instances will affect our understanding of the apparent performance of the SLS algorithm being executed (see Chapter 2.3.6).

[3] T_{run} is usually small in real time applications. For example, if a solution must be available every 1 hour, we cannot afford to have an SLS algorithm that can give us the best solution but in 1 hour 1 minute.

[4] Tuning is not only relevant for SLS, but also applicable to either exact (e.g. [34]) or non-exact algorithms which have some 'tune-able' heuristics. Usually, modifying these 'tune-able' properties can yields different performance. However, we will only discuss the SLS Design and Tuning Problem in this thesis.

picking the most appropriate $M + \phi$ cannot be done *by chance* and that such process will require a form of *intelligence*.

The definition above extends the definition in [6] where the aspects of configuration ϕ are now widened to include previously unconsidered aspect: 'search strategies', which may entails the redesign of the entire SLS algorithm. In Chapter 3.2 below (and in Table 3.1), we propose a new classification for SLS DTP to put into perspective our view of the SLS DTP.

Classification of SLS Design and Tuning Problem

In the literature, the term 'tuning' is often used to refer broad areas: "any action that makes the SLS perform better". To be more precise, we classify SLS DTP into three types (see Table 3.1), according to the number of possible items that require algorithm re-design or further fine-tuning in an SLS algorithm (see Chapter 2.3.4).

SLS Design and Tuning Problem		
Type-1: Calibrating Parameter Values	**Type-2**: Choosing Best Components	**Type-3**: Adding Search Strategies

Table 3.1: The Classification of SLS DTP

Type-1: Calibrating Parameter Values

> Examples: calibrating tabu tenure (TS); perturbation strength (ILS); temperature T (SA); control parameters α, β, exploitation or exploration factor q_0, number of ants M (ACO); population size, recombination probability, mutation probability (GA); etc.

In this 'easiest' type of SLS Design and Tuning Problem, the SLS algorithm has been completely defined (SLS template M and SLS components) and all the algorithm designer needs to do is to set the appropriate exogenous *parameter values*, e.g. setting the tabu tenure for TS, etc. Different parameter values may influence the overall SLS performance.

The challenge is that varying the value of one parameter may affect the optimal setting of the other parameter values since the parameters are often correlated. Furthermore, in many practical situations, the range of parameter values is too large for the algorithm designer to determine their values through trial-and-error.

Type-2: Choosing Best Components

> Examples: choosing the best: local neighborhood (TS/SA/LS); tabu list (TS); perturbation and acceptance criteria (ILS); cooling function (SA); pheromone table (ACO); recombination and mutation operator (GA); etc.

In this type of SLS Design and Tuning Problem, the algorithm designer needs to choose several components that will be used in a particular SLS algorithm, e.g. choosing neighborhood (2/3/k-Opt), tabu list (tabu move/attributes), etc. Typically, each choice of SLS component has its own strengths and weaknesses. Charon and Hudry [10] show that different components have different effects to the performance of SLS algorithm.

Finding the optimal mix of components of the SLS algorithm is often a challenging task as one needs to try substantial number of combinations. We argue that this type of SLS DTP is considered to be more complex than type-1, because choosing appropriate SLS components (this type-2) entails setting appropriate parameter values too (type-1).

Type-3: Adding Search Strategies

> Examples: adding Reactive or Robust Tabu Search strategy (TS); adaptive perturbation (ILS); reheating mechanism (SA); min-max pheromone updates (ACO); diversity preservation (GA); Hybrids; etc.

In this type of SLS Design and Tuning Problem, the algorithm designer needs to design *additional* search strategies to optimize the *run-time dynamics* of the algorithm. These additional search strategies are optional, as the chosen SLS already has some basic search strategy for exploring the fitness landscape. However, good additional search strategies may improve the overall

performance in short runs: steer the search trajectory to more promising fitness landscape regions faster; and in long runs: do some diversification mechanism when the search stagnates.

Unfortunately, search strategies are often problem-specific and deriving the correct ones is tricky even though the number of possible search strategies can only be limited by one's own imagination[5]. The effectiveness of search strategies is also strongly dependent on the correct timings in which they are applied, which in turn introduce more parameters and rules. Also, a search strategy does not always perform what it is intended to perform due to one reason or another as illustrated in various 'failure modes' in [45].

The potential benefits in one hand and the challenges in the other hand have made this type of SLS DTP a rather tedious but yet important task. In the competitive world nowadays, adding good search strategies into our SLS is almost a necessity in order to obtain good performing SLS.

3.3 The Need for a Good Solution

Quotes from Various Researchers

This SLS Design and Tuning Problem is a serious issue. A compilation of some comments from experts in the field highlight both the importance and difficulties of addressing this issue:

- Addressing SLS DTP is itself a scientific endeavor

  > *"The selection of parameter values that drive heuristics is itself a scientific endeavor, and deserves more attention than it has received in the Operations Research literature ..."* — **Barr *et al.* in [4]**

- In the past (mid 1990-ies), it was a pure art rather than science[6]

  > *"Design of a good meta-heuristic remains an art. It depends on the skill and experience of the designer and the empirical computational experiments. In recent literature, there are calls for combining the best features and aspects of each meta-heuristic looking for such unified meta-heuristics. These calls invite more research and empirical analysis."* — **Osman and Kelly (eds) in the preface of [37]**

- Tuning as integral part of the SLS development process

  > *"A quick-and-dirty implementation of an SLS algorithm will give an average performance; but when truly good results and/or faster algorithm are required within tight development time, one should tune his algorithm. Tuning should be viewed as an integral part of the development process. For obtaining a fully functioning algorithm, an SLS needs to be configured: typically some modules need to be instantiated (Type-2, ed) and some parameters (Type-1, ed) need to be tuned."* — **Birattari's PhD Thesis [6]**.

- More time is spent on tuning than designing ...

  > *"There is anecdotal evidence that about* 10% *of the total time dedicated to designing and testing of a new heuristic (or SLS, ed) is spent on development, and the remaining* 90% *is consumed (by, ed) fine-tuning (its, ed) parameters."* — **Adenso-Diaz and Laguna [1]**

- Optimizing SLS may require interaction of multiple components

  > *"Thus, one should keep in mind that the optimization of an Iterated Local Search may require more than optimization of the individual components"* — **(Stützle in Handbook of SLS [15], chapter 11, page 331).**

- That usually adaptive methods are preferred

[5] A recent topic in SLS research is about *hybridization*, in which one SLS is hybridized with other SLS algorithms and/or with other techniques such as linear programming and B & B. While such hybridization strategy can further exploit the beneficial effect of intensification or diversification, it also adds a whole lot other strategies to be tried.

[6] It is now a little bit less art and a little bit more scientific.

"Adaptive perturbation: the behavior of ILS for the QAP and also for other combinatorial optimization problems shows that there is no a priori single best size for perturbation. This motivates the possibility of modifying the perturbation strength and adapting it during the run." — **(idem)**

- It hinders or detrimental to the advancement of SLS research

  — **Hutter *et al.* [24]**

- SLS DTP is not without the danger of 'over-fitting'

  "Good in solving training cases, bad in the actual test cases..."— **Machine Learning [35]**

- There is no one-size fits-all strategy

  — **Wolpert and Macready [46]**

Handling SLS DTP in holistic manner

Any SLS user will face this SLS DTP and they must find the solution: SLS that is carefully designed and configured to attack the underlying COP under its current context.

Formerly, due to the difficulty of the SLS DTP, algorithm designers chose to deal with the type-1 and type-2 problems only. This often results in not so good performance. One may have a good set of components of the SLS algorithm and has all its parameters properly set. But, if the SLS does not conduct an intelligent exploration of the fitness landscape, it will often be outperformed by a dynamic, adaptive, self-correcting, and more intelligent counterpart.

A simple example has been shown by Reactive-Tabu Search [5], where a good search strategy which is able to adaptively adjusts the tabu tenure can outperform the performance of the original, static Tabu Search, on the set of unknown *future* instances — even if the tabu tenure setting of the static Tabu Search is the 'best' over the set of *training* instances.

Our classification in Table 3.1 put this type-3 of SLS DTP into the picture. Ideally, we believe that to obtain the best solution for the SLS DTP, all types of SLS DTP must be addressed properly, in this order: start from type-2 (select the most appropriate SLS components), then type-3 (add search strategies to navigate the search), and then type-1 (set the parameters of the chosen components and strategies).

3.4 Addressing the SLS DTP

In this section, we review several kind of approaches and some real examples in addressing this SLS Design and Tuning Problem:

Blind/Ad-Hoc Tuning

The most naïve way to address SLS DTP is what we termed as blind, brute-force, or ad-hoc tuning. Blind here is that the algorithm designer does not use available tools to aid them in gaining insights. The algorithm designer conducts trial-and-error process to find the best SLS parameter values and/or components. He usually starts by blindly following the suggested parameter values and/or components found in a paper that they read. If it does not work, change here and there a bit and try again. This approach is very tedious and will not work if the COP being attacked is a substantial variant of the version available in the literature, and more severely if there is no paper yet describing how to attack such COP.

Unfortunately, even if this approach is the lousiest, this is also the most common approach in majority of the cases, especially in industry setting where SLS experts are scarce. There are better ways, as shown below.

Brute-Force Tuning using the Power of Computing

Unlike before 1990-ies, computers are now no longer stand alone. More CPU cores are crammed into a single chip. Several computers can share jobs via local networks or Internet. Parallelism and distributed computing are everywhere.

We can take this idea to deal with SLS DTP. Rather than performing rigorous tests to choose which $M+\phi$ is best for a certain COP (typical for stand alone computer), just try those n potential $M+\phi$ on n parallel machines on the same set of COP instances. For every instance, report the best results out of those n machines. Using the same running time t_{run} as in stand alone version, we will likely obtain much better results.

The major drawback is that often we do not have too many n. Typical dual-core processors nowadays allow $n = 2$. Most of the time n is not large. In most cases, we must be able to identify potential $M + \phi$ and not just try all using brute-force method and see which one works.

Self-Correcting Algorithms

Another approach to SLS DTP is to design an SLS algorithm with some kind of self-correcting search strategy. The SLS detects some events during the search (e.g. Tabu Search seems stuck in a local optima) and modify itself automatically (e.g. increase Tabu Tenure to encourage diversification). This way, we do not need to find the best static parameter values (e.g. Tabu Tenure) but let the algorithm adjust its own parameters. The results are generally more robust.

Of course, one needs a way to come up with the proper self-correcting search strategy. This process is classified as type-3 SLS DTP: 'adding search strategy'.

Meta SLS

Yet another approach is to use a meta SLS to configure another SLS. For example: in Evolving ACO [8, 38], a Genetic Algorithm (GA) is used to tune the parameter of the underlying Ants Colony Optimization (ACO) [α, β, q_0, M]. These ACO parameters are embedded in GA's chromosome. So, GA runs ACO several times to attack the underlying COP. ACO results are used by GA to determine the next population of ACO parameters. This process is called 'Meta-evolution'.

While this technique is good and liberates us from setting any parameter, this process is (very) slow. Major part of the limited running time t_{run} is 'wasted' to 'evolve' the main SLS. Given the same running time limit t_{run}, specialized algorithm will be likely the winner.

General Approaches: Black-Box and White-Box

Other than the approaches above, there are several general approaches to address the SLS Design and Tuning Problem. General means that these approaches are virtually applicable for most cases: for most SLS algorithms applied to most COPs. We classify these general approaches into two major types: black-box versus white-box approaches. The details of the classification are shown in the Table 3.2.

3.5 Black-Box Approaches

Simply said, black-box approaches are direct extensions of blind tuning, more generic than meta SLS, and utilizes computing power in a better way. We present several examples below.

CALIBRA

Adenso-Diaz and Laguna [1] proposed a tool to automatically calibrate parameter values when given pre-defined ranges. It works by iteratively calling the target algorithm with various set of parameter values and then uses the objective value feedbacks to determine which set of parameter values should be used in the next iteration. CALIBRA uses Taguchi's fractional factorial design to keep the number of parameter values being tried to be within acceptable limit. Iteratively, CALIBRA can narrow down the range of the algorithm parameters until the values converge to a 'local minima' in the configuration space. After the maximum number of iterations elapsed,

	Black-Box Approaches	White-Box Approaches
Definition	• Treat SLS as 'black-box'. Usually in form of automated tools that systematically search for the best parameter values or combination of SLS components given set of (big) initial configurations. • Human role is to provide initial configurations, with minimal feedbacks, if any. Human may decide to restart the black-box tool using different initial configurations upon gathering data from previous configuration attempts.	• Open up the 'box' to allow the algorithm designer to inspect the inner-working of the algorithm and to assist in designing a better SLS algorithm. • Require (heavy) collaboration with human, strong feedback links that must be analyzed using human intelligence.
Strengths	• Can relieve the burden of addressing parts of the SLS DTP from human, especially type-1 and type-2.	• Can address type-1, type-2, and especially type-3 of SLS DTP. • Allow for possible human creativity, innovations, or inventions. • Can give insights on understanding the SLS or the COP.
Weaknesses	• Do not allow for human creativity, innovations, or inventions. • Have difficulty in handling type-3 of SLS DTP. • Often need to try too many possibilities, thus in tight development time, they can only run/test each configuration in a relatively short period of time. • This approach cannot be used to debug or to improve the underlying design of the SLS algorithm.	• Human still need to do substantial works. • Human must understand the behavior of the SLS. • The results may be inconsistent as different users design and tune the SLS algorithm differently. • The time required to deal with the SLS DTP is also a variable as it depends on the expertise of the user. • Not suitable for fine tuning parameter values (type-1 SLS DTP).

Table 3.2: Black-Box versus White-Box Approaches

CALIBRA will return the best set of parameters found so far. This way, it manages to solve the type-1 and type-2 of SLS DTP[7] quite reasonably.

CALIBRA has limitations. The current version (2006) can only tune up to 5 parameters, the other parameters must be fixed to 'appropriate values'. The need to supply initial range is also problematic when one does not know a good starting range for certain parameters, forcing him to resort to trial and error approach or using default values only. Furthermore, CALIBRA is not specifically designed to address type-3 of SLS Design and Tuning Problem.

CALIBRA (2006) is available at `http://coruxa.epsig.uniovi.es/~adenso/file_d.html`.

F-Race

Birattari [7, 6] proposed a racing algorithm, a method that was previously known in the machine learning community, to address SLS Design and Tuning Problem. The racing algorithm (F-Race), paraphrased from his work, can be summarized as follows: First, feed F-Race with a (possibly large) set of candidate configurations. F-Race will estimate the expected performance of the candidate configurations in an incremental way and discard the worst ones as soon as sufficient statistical evidence[8] is gathered against them. This allows a better allocation of computing

[7] CALIBRA only deals with numeric values, but one can still use it to configure SLS components by implementing the SLS using if-else rules. When CALIBRA selects a certain numeric constant, a certain SLS component mapped to this numeric constant is activated. See Chapter 2.3.5 for details of such SLS implementation.

[8] Birattari used Friedman non-parametric tests for this purpose. For more details about statistical tests like this, please see [16].

power because rather than wasting time evaluating low-performance configurations; the algorithm focuses on the assessment of the better ones. As a result, more data is gathered concerning the configurations that are deemed yielding better results, and eventually a more informed and sharper selection is performed among them. Finally, the last configuration is declared as the winner (best) configuration. This process is very much analogous with the real life racing.

The number of possible configurations to be tested can be very large, thus by not trying every possible configuration blindly, F-Race is much better than systematic brute force try-all approach. F-Race is classified as type-1 and type-2 SLS DTP solver as it can be used to find good parameter values and proper combination of components of the algorithm simultaneously.

However, F-Race also has several inherent limitations, which arise from the fact that it is a black-box approach. Similar to CALIBRA, F-Race is unable to help the algorithm designer to give solution beyond the best configuration found in the set of possible configurations initially supplied to the tuning algorithm. One should also be aware of the 'combinatorial explosion' of the number of configurations to be tried, as it will require enormous computation time that may possibly exceeding the maximum allowed development time. Hence, to keep the size of initial set of configurations small, the algorithm designer must intelligently decide which should be included in the set, a process that preferably not be done *blindly*.

Racing approach is quite accepted as promising way to address type-1 and type-2 SLS DTP, as shown by the usage of racing approach in more recent SLS works [47, 48].

F-Race (2005) is available at `http://code.ulb.ac.be/iridia.activities.software.php`.

3.6 White-Box Approaches

If black-box approach favors machine, white-box approach favors human intelligence. Algorithm designers have devised various techniques to assist them in understanding their SLS algorithms' behavior in order to adjust the correct parts of their SLS algorithms. The result is a better-designed SLS algorithm (e.g. the ones with self-correcting strategies).

White-box approaches are either in form of statistical analysis (non-visualization) or information visualization techniques. They are discussed below.

Descriptive Statistics

Descriptive statistics are good for summarizing the data. The most widely used statistic for analyzing SLS is **'Solution Quality'**, e.g. the SLS performance is measured against benchmark instances[9] which can be obtained from benchmark libraries (e.g. TSPLIB [40, 44] for TSP, QAPLIB [9, 39] for QAP, etc) or created using exact algorithms for sufficiently small instances.

'Robustness' is another desirable statistic for measuring the performance of SLS algorithms with stochastic elements. Since the SLS algorithm behaves non-deterministically by selecting different search trajectory on different runs, robustness analysis measures the degree of variation of the best-found solution found by the algorithm. Obviously, the more robust/consistent the algorithm is, the better it is.

Solution quality and robustness can be visualized like Figure 3.1. A robust (red) and not robust (blue) algorithm performance can be easily seen.

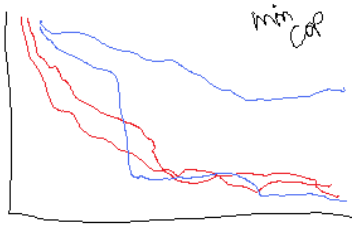

Figure 3.1: Solution Quality Visualization

[9]When one does not know the global optima for a particular instance, he can use different SLS with different configurations, record the best known value so far, and publish his best result to public domain. This is the standard that needs to be improved over time by himself or by other researchers.

'Speed or Running Time' is the main advantage of SLS algorithms over exact algorithms. As such, an SLS algorithm should be fast w.r.t the exact algorithm counterpart. Speed can be measured as the time to reach the best-found solution. Currently, rather than using simple statistics averaged over runs and problem instances, a more realistic picture can be obtained by characterizing the **Run Time/Length Distribution (RTD/RLD)** statistic of SLS algorithms [22, 42].

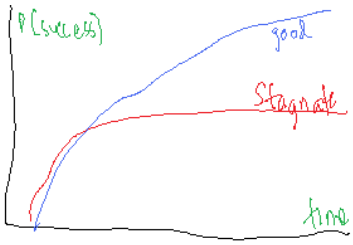

Figure 3.2: RTD Visualization

'Properties of the Fitness Landscape' of the COPs, such as the distribution and number of local optima, the existence of 'big valleys', landscape ruggedness, the existence of plateau region, etc, are known to affect the difficulty of the COP instance which in turn influence the performance of the SLS algorithms. Even though the fitness landscapes of COP instances can be enormously large and exploring the entire search space is almost unlikely, one may still gather crucial statistical properties of the fitness landscape.

Techniques like **Fitness Distance Correlation (FDC)** analysis [25, 26, 13, 32, 23, 45] are used to measure whether such properties can be exploited. The result of such analysis, if interpreted and reasoned correctly, may yield interesting discoveries that can be exploited to improve the design of the SLS algorithms.

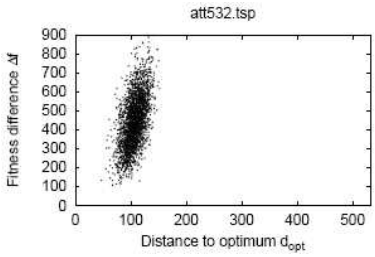

Figure 3.3: FDC Scatter Plot

For example, by knowing that TSP has 'Big Valley' property, we can design an SLS algorithm to concentrate on the region near the current best-found solution, by knowing that the fitness landscape is very smooth, then any SLS algorithm that can do hill climbing will get relatively 'good' solution, by knowing that the fitness landscape is very rugged, we can design SLS algorithm to perform stronger escape mechanism.

Statistical methods can be used to address all types of the SLS DTP, even though less suitable to address type-1. However, this process is not straightforward. Knowing the statistical information about the fitness landscape of a COP is a necessary but not sufficient condition to design a good SLS for that COP. A significant amount of human effort is still required to reason on the facts found using statistical analysis before a good solution for SLS DTP can be produced. In the context of SLS DTP, this lengthy process is undesirable due to tight development time. We also argue that without a proper computer-aided tool, it is difficult to generate the required solution within tight development time, with merely statistical data.

Information Visualization

If we are only interested with a summary, then statistical analysis above may already be sufficient. However, such summary hides trends, patterns, discontinuities, outliers, or anomalies in the data. We usually also need to see the details. Statistics alone cannot succinctly describe the detailed search behavior.

However, the amount of detailed information collected during SLS algorithm execution can be very large, depending of what is being recorded. Thus, it is necessary to present the data in an intuitive form which can be effectively analyzed and interpreted by human. Fortunately, human visual perception system has very high information bandwidth. Human can detect patterns with perhaps just one look at an image. The science to exploit human visual perception is given a name: Information Visualization. This technique is used to present the detailed information about SLS runs and to enhance simple textual output of statistical analysis into its visual counterparts.

Information visualization fans will agree understanding (statistical) data is generally easier via visualization rather than via numbers. Moreover, information that can be derived from dynamic visualization is richer than condensed statistical information. Having additional visualizations may complements the existing statistical techniques.

Here, we review some examples of visualizing SLS algorithms.

Human-Guided Search

Human-guided search (HuGS) tries to utilize human visual perception and intelligence by providing the user with a good visualization and interaction tool. HuGS presents the *problem-specific* visualization of the current solution (e.g. TSP tours, etc) and allow user to control the search (e.g. focusing the SLS on subset of edges only, etc). This may works because in general we know the ingredients of good solutions of the COP. Perhaps, our guidance may be able to assist the algorithm to obtain good results quicker.

Figure 3.4: Human Guided Search

Research on interactive man-machine optimization can be found in as early as late 1960-ies [33, 30]. Recently, this line of work is re-surfaced in Mitsubishi Electric Research Laboratory (MERL)'s projects: [3, 2, 29, 28, 41, 11].
The drawbacks: guiding the search for a prolonged period of time is tedious and sometimes we do not know what to do to steer the search. Thus, the effectiveness of HuGS is limited by the stamina, patience, and intuition of the human user.

In its original intention, human-guided search is not meant to be used as an approach for addressing the SLS DTP. However, the search strategies derived when guiding the search can be made permanent (implemented as part of the search strategy of the SLS algorithm).

Visualization of Search Behavior

Syrjakow and Szczerbicka [43] proposed a visualization for a simple optimization problem with **2 decision variables only** (see Figure 3.5). The problem size is quite small so that the entire fitness landscape can be fully computed. The search positions are then animated in this visualization. This is intuitive, but this approach is limited for 'toy problems' only. In practice, COP solutions are n-dimensional, which require some transformations to be displayed in 2-D screen. Constructing the complete fitness landscape is almost impossible for $\mathcal{NP}$-hard COP.
Kadluczka [27] proposed a generic visualizer that is slightly better than the one above. The authors proposed a mapping scheme for N-dimensional objects to 2-D space which can be displayed on the screen. By plotting the positions of the N-dimensional solutions in 2-D space, one can approximately identify which search space has/has not been explored by the SLS search algorithm. This information can be used as a guidance to improve the algorithm.

In Figure 3.6, we observe that starting from the points denoted with S, we can trace the behavior of the algorithm in their attempts to reach various local optimal solutions G. By drawing such

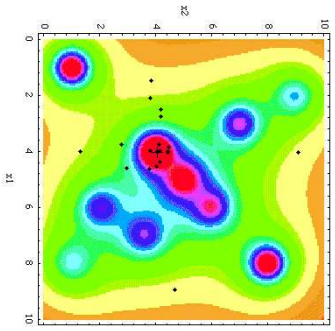

Figure 3.5: Straightforward Visualization of COP with only 2 Decision Variables

visualization, we can gain insights of the search performance. Here, we can see that the search space covered by the search algorithm and conclude that Tabu Search (blue) manages to visit search space more thoroughly than simple Local Search (orange). The several white space regions in the visualization imply that the regions are not yet explored, which imply that diversification mechanisms may be required.

The limitation of this approach is that the huge gap in size between of the *exponential* search space ($O(k^n)$ or $O(n!)$) and the *polynomial* screen space renders such visualization inappropriate for larger values of n. Furthermore, the static visualization adopted in this work (see Figure 3.6) does not convey the run-time dynamic behavior of SLS search well.

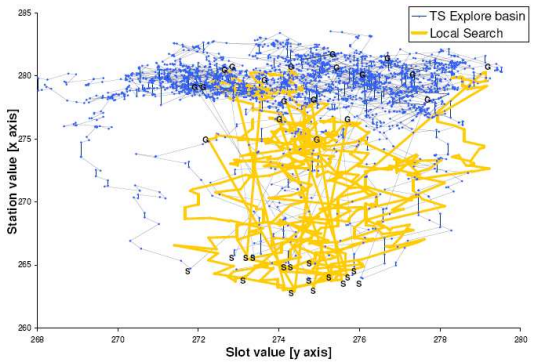

Figure 3.6: N-to-2-Space Mapping. See text for details.

3.7 Comparison of the Approaches

Out of those approaches discussed in this chapter, some are better than the other in some aspects, but just no overall winner.

Clearly, blind tuning is not advise-able. There are better ways than this laborious manual trial and error. Utilizing computing power blindly is better than manual effort but yet it is not the best. Having SLS that can correct itself is good, but how to come up with such SLS in the first place. The idea of having 'parameter-less' SLS via Meta SLS is good, but it is slow. Finally, between the general approaches of black-box and white-box, it is debate-able which one is better.

In general, black-box approaches are good for fine-tuning, selecting the best configuration in a relatively small initial configuration set. Doing this manually is very tedious. If the COP instances being attacked is quite homogeneous, using black-box approaches alone may be adequate to fine-tune the SLS algorithm to perform well. However, if black-box approaches improve the SLS performance by choosing better configurations, they do so without explaining why the chosen configurations work. This is not good in advancing the SLS research.

White-box approaches naturally allow more customizations or innovations to the SLS algorithm. This is more suitable for choosing appropriate SLS component and search strategies as determining these requires insight about both the COP and the SLS. This flexibility suits COP which have instances that are quite heterogeneous as the SLS must be adapted to different types. However,

white-box approach is just a tool which will not actually improve the quality of the SLS algorithm unless human do something with the obtained insights.

In Table 3.3, we provide our *subjective* view of the differences of each approaches with respect to our tool Viz (see Chapter 4 and 6) as the basis of comparison.

Techniques	**A-Cal**	**B-Race**	**C-Stat**	**D:HuGS**	**E-Visu**	**F-Viz**
Type of Method	Black	Black	White	White	White	Integrated
Can address type-1?	Easy	Easy	Hard	Hard	Hard	Average
Can address type-2?	Average	Easy	Hard	Hard	Hard	Easy
Can address type-3?	N/A	N/A	Hard	Average	Average	Easy
Ease of Usage	Easy	Easy	Hard	Average	Average	Average

Table 3.3: Comparison of Several Existing Approaches

Legends: **A**: CALIBRA, **B**: F-Race, **C**: Statistical Analysis, **D**: Human-Guided Search, **E**: Visualization of Search Algorithm Behavior, **F**: Viz.

3.8 Issues in Addressing SLS DTP

Before we end this chapter, it is worth to mention important issues that often violated when one is dealing with the SLS Design and Tuning Problem:

Danger of using black-box approaches only

There is an obvious weakness of using black-box approaches only. Because we do not investigate the actual behavior of our algorithm, we may end up believing that we have improved the 'wrong/faulty' algorithm. For example, a faulty algorithm is passed to a black-box tuning tool. No matter which parameter values or SLS components selected by the tuning tool, it is still wrong.

Development time t_{dev} is limited

In practice, we usually have *tight* development time while building the SLS algorithm. This is often violated in the literature when researchers devote a large amount of time, exceeding the typical normal development time allowed in real life, to develop their algorithm. A solution for SLS DTP must be fast, easy to use.

Running time t_{run} is limited

If we run an SLS for a very long time, it may luckily reach very good solutions and we may mistakenly conclude that the SLS is good (ceiling effect). If all implementations are reporting good results, it is hard to differentiate the results.

Unfair comparison

We also should not compare results unfairly. Unfair comparisons will favor our new algorithm for winning the comparison. The list below shows potential violations:

1. Comparing our algorithm against quick-and-dirty implementations of state-of-the-art algorithms.
2. Implementing state-of-the-art algorithms using different (slower) programming languages and run it in (slower) machines.
3. Do not implement the state-of-the-art algorithms but rather comparing the results obtained by our algorithms on our current and fast computers with those published in the literature long time ago and possibly obtained using much slower computers at that time.

Over-fitting to training instances

Sometimes, after spending enormous effort, researchers managed to tune their method so well that it can solve a particular set of benchmark instances with a very high quality. But frequently, these algorithms usually are not general enough to solve the remaining instances in the benchmark set, thus classified as 'over-fitting/over-tuning' [12], see also [14]. The discussion of No Free Lunch theorem [46], although still open, also suggests that if an algorithm works very well on a particular test instances, it will more likely that it may perform poorer on the remaining instances.

Researchers should not conclude that their algorithm works well for a particular COP if it only works for the training instances used for fine-tuning their algorithm, but rather we should separate clearly the training instances and test instances, as those practiced in machine learning community [6]. An implementation can only be considered as good if after tuned, it works well in general for any future (unknown) instances of the COP.

To reduce the 'over-fitting' issue, we may try using larger training instances will logically force the tuning procedure to select a less specialized and more general configurations. However, sometimes it may be beneficial to partition the set of instances of the COP into separate classes if we know clearly the distinguishing characteristics of each class. In that case, it may be better to have two SLS algorithms that are each 'over-fitted' to certain class of instances.

3.9 Summary

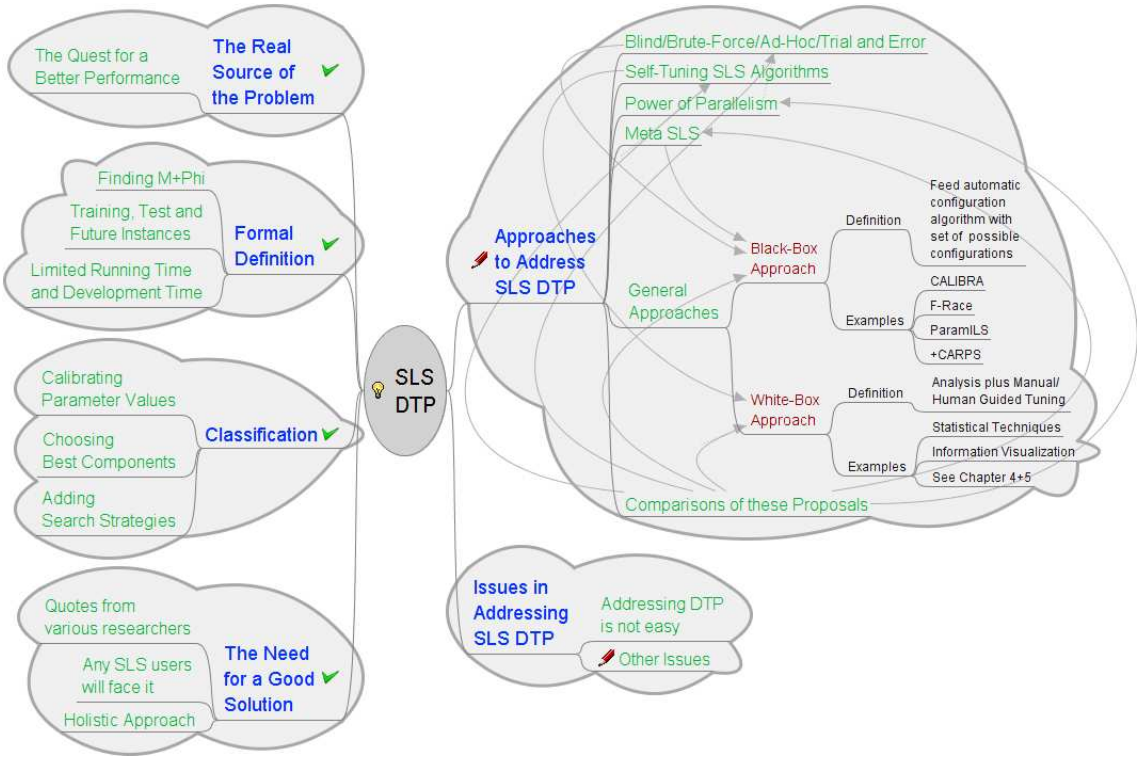

Figure 3.7: Chapter Overview

1. Humans always want 'a better one', including a better SLS algorithm to attack their COP.
2. Although it can be easy to come up with a working SLS algorithm to attack a particular COP, designing and tuning the implementation of such algorithm to achieve consistently good results in short amount of development and running time is not so straightforward. We call this problem: the SLS Design and Tuning Problem.
3. The term SLS DTP is big. To be precise, we classify it into three types:
   (a) Calibrating parameter values

(b) Choosing best components, and

(c) Adding search strategies

4. SLS DTP is unavoidable as we cannot change the COP being attacked so that it suits our SLS algorithm. We must be the one fighting this SLS DTP in order to create a good SLS algorithm for the given COP. Experts in the field agree with the importance of having this issue addressed.

5. There are several approaches to address the SLS DTP: blind approach, brute-force using computer strengths, using self-correcting algorithms, meta SLS, or using general approaches.

6. General approaches to address the SLS DTP are classified into two: black-box and white-box approaches, each has their own strengths and weaknesses. We have elaborated and reviewed several examples in this chapter.

7. Despite many proposed approaches, there is no clear winner yet. A better way is still very much required.

3.10 Further Discussions

In the literature, we observe various approaches to address this SLS Design and Tuning Problem, with no approach is clearly superior than the others in addressing all types of SLS DTP.

In Chapter 4, we propose a white-box information visualization technique (FLST visualization). With this FLST visualization, one can better understand their SLS algorithm and use their intelligence to help finding part of the solution for the SLS DTP. In Chapter 5, we go one more step by integrating both the white-box and black-box approaches to form a stronger methodology.

Bibliography

[1] Belarmino Adenso-Diaz and Manuel Laguna. Fine-tuning of Algorithms Using Fractional Experimental Designs and Local Search. *Operations Research*, 54(1):99–114, 2006.

[2] David Anderson, Emily Anderson, Neal Lesh, Joe Marks, Brian Mirtich, David Ratajczak, and Kathy Ryall. Human-Guided Simple Search. In *17th National Conference on Artificial Intelligence*, pages 209–216, 2000.

[3] David Anderson, Emily Anderson, Neal Lesh, Joe Marks, Ken Perlin, David Ratajczak, and Kathy Ryall. Human-Guided Greedy Search: Combining Information Visualization and Heuristic Search. In *Workshop on New Paradigms in Information Visualization and Manipulation*, pages 21–25, 1999.

[4] Richard S. Barr, Bruce L. Golden, James P. Kelly, Mauricio G.C. Resende, and W.R. Stewart. Designing and Reporting on Computational Experiments with Heuristic Methods. *Journal of Heuristics*, 1:9–32, 1995.

[5] Roberto Battiti and Giampietro Tecchiolli. The Reactive Tabu Search. *ORSA Journal on Computing*, 6(2):126–140, 1994.

[6] Mauro Birattari. *The Problem of Tuning Metaheuristics as seen from a machine learning perspective*. PhD thesis, University Libre de Bruxelles, 2004.

[7] Mauro Birattari, Thomas Stützle, Luis Paquete, and Klaus Varrentrapp. A Racing Algorithm for Configuring Metaheuristics. In *Langdon, William B. et al. (eds). Genetic and Evolutionary Computation Conference*, pages 11–18. Morgan Kaufmann, 2002.

[8] Hozefa M. Botee and Eric Bonabeau. Evolving Ant Colony Optimization. *Advanced Complex Systems*, 1:149–159, 1998.

[9] Rainer E. Burkard, Stefan E. Karisch, and Franz Rendl. A quadratic assignment problem library. *European Journal of Operational Research*, 55:115–119, 1991.

[10] Irène Charon and Olivier Hudry. Mixing Different Components of Metaheuristics. In *Meta-Heuristics: Theory and Applications*, pages 589–603. Kluwer Academic Publishers, 1996.

[11] Markus Chimani, Neal Lesh, Michael Mitzenmacher, Candace L. Sidner, and Hidetoshi Tanaka. A Case Study in Large-Scale Interactive Optimization. *Artificial Intelligence and Applications*, pages 24–29, 2005.

[12] Emanuel Falkenauer. On Method Overfitting. *Journal of Heuristics*, 4:281–287, 1998.

[13] Cyril Fonlupt, Denis Robilliard, Philippe Preux, and El-Ghazali Talbi. Fitness Landscapes and Performance of Meta-heuristics. In *Meta-Heuristics - Advances and Trends in Local Search Paradigms for Optimization*, pages 255–266. Kluwer Academic Publishers, 1999.

[14] Ian Gent. A Response to 'On Method Overfitting'. *Journal of Heuristics*, 5:108–111, 1998.

[15] Fred Glover and Gary A. Kochenberger. *Handbook of Metaheuristics*. Kluwer Academic Publishers, 2003.

[16] Frederick J. Gravetter and Larry B. Wallnau. *Statistics for the Behavioral Sciences*. Thomson Wadsworth, seventh edition, 2007.

[17] Steven Halim and Hoong Chuin Lau. Tuning Tabu Search Strategies via Visual Diagnosis. In *Meta-Heuristics: Progress in Complex Systems Optimization*, pages 365–388. Kluwer Academic Publishers, 2007.

[18] Steven Halim and Roland Hock Chuan Yap. Designing and Tuning SLS through Animation and Graphics: an Extended Walk-through. In *Engineering Stochastic Local Search*, pages 16–30, 2007.

[19] Steven Halim, Roland Hock Chuan Yap, and Hoong Chuin Lau. Visualization for Analyzing Trajectory-Based Metaheuristic Search Algorithms. In *European Conference on Artificial Intelligence*, pages 703–704, 2006.

[20] Steven Halim, Roland Hock Chuan Yap, and Hoong Chuin Lau. Viz: A Visual Analysis Suite for Explaining Local Search Behavior. In *19th User Interface Software and Technology*, pages 57–66, 2006.

[21] Steven Halim, Roland Hock Chuan Yap, and Hoong Chuin Lau. An Integrated White+Black Box Approach for Designing and Tuning Stochastic Local Search. In *Principles and Practice of Constraint Programming*, pages 332–347, 2007.

[22] Holger H. Hoos. *Stochastic Local Search: Model, Analysis, and Applications*. PhD thesis, Technical University Darmstadt, Germany, 1999.

[23] Holger H. Hoos and Thomas Stützle. *Stochastic Local Search: Foundations and Applications*. Morgan Kaufmann, 2005.

[24] Frank Hutter, Holger H. Hoos, and Thomas Stützle. Automatic Algorithm Configuration based on Local Search. In *National Conference on Artificial Intelligence*, 2007.

[25] Terry Jones. *Evolutionary Algorithms, Fitness Landscapes and Search*. PhD thesis, The University of New Mexico, Albuquerque, New Mexico, 1995.

[26] Terry Jones and Stephanie Forrest. Fitness Distance Correlation as a Measure of Problem Difficulty for Genetic Algorithms. In *6th International Conference on Genetic Algorithms*, pages 184–192, 1995.

[27] Marcin Kadluczka, Peter C. Nelson, and Thomas M. Tirpak. N-to-2-Space Mapping for Visualization of Search Algorithm Performance. In *International Conference on Tools with Artificial Intelligence*, pages 508–513, 2004.

[28] Gunnar W. Klau, Neal Lesh, Joe Marks, and Michael Mitzenmacher. Human-Guided Tabu Search. In *National Conference on Artificial Intelligence (AAAI)*, pages 41–47, 2002.

[29] Gunnar W. Klau, Neal Lesh, Joe Marks, Michael Mitzenmacher, and Guy T. Schafer. The HuGS Platform: A Toolkit for Interactive Optimization. In *Advanced Visual Interfaces, May 2002, Trento, Italy*, 2002.

[30] P. Krolak, W. Felts, and G. Marble. A Man-Machine Approach Toward Solving The Traveling Salesman Problem. *Communications of the ACM*, 14(5):327–334, 1971.

[31] Hoong Chuin Lau, Wee Chong Wan, and Steven Halim. Tuning Tabu Search Strategies via Visual Diagnosis. In *6th Metaheuristics International Conference*, pages 630–636, 2005.

[32] Peter Merz. *Memetic Algorithms for Combinatorial Optimization: Fitness Landscapes & Effective Search Strategies*. PhD thesis, University of Siegen, Germany, 2000.

[33] Donald Michie, J.G. Fleming, and J.V. Oldfield. A Comparison of Heuristic, Interactive, and Unaided Methods of Solving a Shortest-route Problem. In *Michie, Donald (eds). Machine Intelligence series 3*, pages 245–256. Edinburgh University Press, 1968.

[34] Steven Minton. Automatically Configuring Constraint Satisfaction Programs: A Case Study. *Constraints*, 1(1/2):7–43, 1996.

[35] Tom Mitchell. *Machine Learning*. McGraw Hill, 1997.

[36] Dagmar Monett-Diaz. +CARPS: Configuration of Metaheuristics Based on Cooperative Agents. In *International Workshop on Hybrid Metaheuristics*, pages 115–125, 2004.

[37] Ibrahim H. Osman and James P. Kelly. *Meta-Heuristics - The Theory and Applications.* Kluwer Academic Publishers, 1996.

[38] M.L. Pilat and T. White. Using Genetic Algorithms to optimize ACS-TSP. In *Ant Algorithms, Third International Workshop, ANTS 2002*, pages 282–287, 2002.

[39] QAPLIB. Quadratic Assignment Problem Library. http://www.seas.upenn.edu/qaplib.

[40] Gerhard Reinelt. TSPLIB - A Traveling Salesman Problem Library. *ORSA Journal on Computing*, 3:376–384, 1991.

[41] Stacey D. Scott, Neal Lesh, and Gunnar W. Klau. Investigating Human-Computer Optimization. In *Conference on Human Factors in Computing Systems*, pages 155–162, 2002.

[42] Thomas Stützle and Holger H. Hoos. Analyzing the Run-Time Behavior of Iterated Local Search for the TSP. In *3rd Metaheuristics International Conference*, pages 449–453, 1999.

[43] Michael Syrjakow and Helena Szczerbicka. Java-based Animation of Probabilistic Search Algorithms. In *International Conference on Web-based Modeling & Simulation*, pages 182–187, 1999.

[44] TSPLIB. Traveling Salesman Problem Library. http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95.

[45] Jeal-Paul Watson. On Metaheuristics "Failure Modes". In *6th Metaheuristics International Conference*, pages 910–915, 2005.

[46] David H. Wolpert and William G. Macready. No Free Lunch Theorems for Optimization. *IEEE Transactions on Evolutionary Computation*, 1:67–82, 1997.

[47] Bo Yuan and Marcus Gallagher. Statistical Racing Techniques for Improved Empirical Evaluation of Evolutionary Algorithms. In *8th International Conference in Parallel Problem Solving from Nature (PPSN), Birmingham, UK*, pages 172–181, 2004.

[48] Bo Yuan and Marcus Gallagher. A Hybrid Approach to Parameter Tuning in Genetic Algorithms. In *IEEE International Conference on Evolutionary Computation*, 2005.

[49] Zondervan. *New International Version (NIV) Holy Bible.* Zondervan, 1978.

Index