AN INTEGRATED WHITE+BLACK BOX APPROACH FOR DESIGNING AND TUNING STOCHASTIC LOCAL SEARCH ALGORITHMS

STEVEN HALIM

B.Comp.(Hons.), NUS

A THESIS SUBMITTED

FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTER SCIENCE

NATIONAL UNIVERSITY OF SINGAPORE

2009

Acknowledgements

Though this thesis is the author's personal work, it would not have been completed without the academic, spiritual, or moral supports from many individuals.

First and foremost, I want to thank God the creator; Jesus Christ the savior; and Holy Spirit the companion, for allowing me to live in this world, blessing me with knowledge, and guiding me throughout my PhD candidature.

Next, I would like to express my heartfelt thanks to my main supervisor Dr Roland Yap. You are a critical reviewer of our works, very meticulous, diligent, the first filter for all our research works. I am motivated by your passion for research. Thanks for your interest to develop Viz, especially the part for SLS visualization though it was *not yet* one of your research areas before.

I also want to thank to my second supervisor Dr Lau Hoong Chuin, who was formerly my main supervisor before you moved to SMU mid-2005. Thank you for introducing me into this field of SLS research – which I eventually interested in. If now I can juggle multiple projects simultaneously, it is partly due to your training.

Next, I thank my PhD thesis committee: Dr Martin Henz, Dr Leong Hon Wai, and Dr Thomas Stützle for reading and giving valuable comments to improve this thesis.

I do not *and will not* forget the two most important people in my life, my father Tjoe Tjie Fong and my mother Tan Hoey Lan. Papa, Mama, thanks for raising me up since I was a baby until now. Without your love and sacrifices so far, I would not have been able to be what I am today.

Next, I want to thank Felix Halim, my younger brother, who joined me in SoC, NUS at the later phase of my PhD candidature. I am glad to have someone at home with whom I can discuss research. We even published a paper together [58], perhaps the first of more to come?

A list of people comes next: My former colleagues: Wan Wee Chong, thank you for your guidance during the early years of my PhD candidature, and Xiao Fei, nice to work with you in the later part of my PhD candidature. And for those who cannot be named one by one but contributed in one way or another: my bros and sistas, ex-housemates @ Flynn Park, admin/IT people at SoC, etc, thank you ...

Lastly, reserved for a significant individual in my life, I want to thank my wife: Grace Suryani Tioso. Your words of encouragements and prayers have supported me in going through this PhD candidature. Truthfully, I cannot recount the number of cards/e-cards/SMSes/calls/words that you have used for this supporting role of yours. I am glad to have you as my wife. Thank you dear. I love you very much.

Contents

Summary

This thesis addresses two related Stochastic Local Search (SLS) engineering problems: the **Design** and **Tuning Problem** (DTP). The SLS DTP can be informally defined as a **meta-level** problem of finding a good SLS algorithm which has been tuned for a given class of Combinatorial Optimization Problems (COP). The problem, which this thesis addresses, is a systematic methodology for dealing with SLS DTP which is effective for obtaining better performing SLS algorithms.

Current approaches to address SLS DTP can be classified into **white-box**: analysis of the SLS algorithm and/or the COP being attacked; or **black-box**: automated tuning algorithms that aim to get the best SLS configuration given an initial configuration set. These existing approaches have their strengths and limitations, yet they do not solve the SLS DTP well enough.

A novel contribution of this thesis is a *generic* white-box **Fitness Landscape Search Trajectory** (FLST) visualization. This visualization is designed to allows the algorithm designers to investigate the n-dimensional fitness landscape structure of the COP being analyzed in 2-D. There are obviously visualization errors by using 2-D to show n-dimensional fitness landscape. However, we are able to quantify the errors and provide mechanism for users to identify the errors. We show in this thesis that even with such inherent visualization errors issue with this FLST visualization, the users can still use it to develop insights on what should be a good search strategy for exploring the given fitness landscape, as well as to observe how his current SLS algorithm behaves on that fitness landscape. This enables the algorithm designer to design the SLS algorithm in a more intuitive manner than existing white-box approaches.

The resulting SLS algorithm still needs to be fine-tuned, and we propose an **Integrated White+Black Box Approach** (IWBBA) in which we first start with the white-box FLST visualization above, improve the SLS algorithm, and then pass the implementation of the SLS algorithm to be fine-tuned using black-box approaches, stepping up its performance more. The insights gained from the previous white-box step will likely have pruned the possible configuration set, easing and indirectly improving the performance of the black-box tuning algorithm.

To implement this integrated approach, we have built an SLS visualization and engineering tool called **Viz**. We have successfully applied IWBBA using Viz to develop and improve several SLS algorithms from the literature: Iterated Local Search (ILS) for the Traveling Salesman Problem (TSP), Robust Tabu Search (Ro-TS) for the Quadratic Assignment Problem, and most notably: Tabu Search (TS) for the Low Autocorrelation Binary Sequence (LABS) problem where we obtained state-of-the-art results.

List of Tables

List of Figures

List of Abbreviations

a.k.a : also known as
ACO : Ants Colony Optimization
AI : Artificial Intelligence
AP : Anchor Point
AS : Algorithm Specific

B&B : Branch and Bound
B&C : Branch and Cut
BF : Best Found
BK : Best Known

CAPTCHA : Completely Automated Public Turing Test to Tell Computers and Humans Apart
COP : Combinatorial (or Constraint) Optimization Problem

DTP : Design and Tuning Problem

EB : Event Bar
EW : Experiment Wizard

FDC : Fitness Distance Correlation
FL : Fitness Landscape
FLST : Fitness Landscape Search Trajectory

GA : Genetic Algorithm
GO : Global Optima (a.k.a Global Maxima/Minima)
GUI : Graphical User Interface

HCI : Human Computer Interaction

ILS : Iterated Local Search
IWBBA : Integrated White+Black Box Approach

LABS : Low Autocorrelation Binary Sequence
LO : Local Optima (a.k.a Local Maxima/Minima)
LS : Local Search

MDF : Metaheuristics Development Framework

N : Neighborhood
NFL : No Free Lunch
$\mathcal{NP}$: Nondeterministic Polynomial

OR : Operations Research
OV : Objective Value (a.k.a Fitness)

$\mathcal{P}$: Polynomial

PS : Problem Specific

QAP : Quadratic Assignment Problem

REM : Reverse Elimination Method
RLD : Run Length Distribution
RTD : Run Time Distribution
Re-TS : Reactive Tabu Search
Ro-TS : Robust Tabu Search

SA : Simulated Annealing
SIMRA : Single Instance Multiple Runs Analyzer
SLS : Stochastic Local Search
SC : Search Coverage
ST : Search Trajectory
S-TS : Strict Tabu Search

TS : Tabu Search
TSP : Traveling Salesman Problem
TT : Tabu Tenure (TS)

VLSN : Very Large Scale Neighborhood

w.r.t : with respect to

Chapter 1

Introduction

A journey of a thousand miles must begin with a single step.
— **Lao Tzu**

In this introductory chapter, we present the motivation that inspires the research in this thesis and the summary of our contributions to this field.

1.1 The SLS Design and Tuning Problem

In this thesis, the difficult computational problems that we are dealing with belong to the class of $\mathcal{NP}$-complete Combinatorial Optimization Problems (COPs[1]). Here, we are interested to find the best out of (extremely) many possible *combinatorial* solutions of a given COP. The 'best' is defined by the user's objective function.

COPs are found in *many* practical applications (see Appendix A) and thus solving these COPs efficiently is a necessity. The most naïve algorithm to solve COPs is to enumerate all possible solutions and simply pick the best (optimal) one. However, these naïve enumeration algorithms are impractical, rendering any attempts for solving these COPs using them ineffective especially for large instances.

Solving $\mathcal{NP}$-complete COPs (Section 2.2 - 2.3)

Computer scientists have devised various *exact* algorithms that are better than the naïve enumeration algorithms. One example of such exact algorithms is the 'Branch & Bound' algorithm. Notwithstanding their capabilities, these exact algorithms still run into the computational intractability limits of such problem.

Mainly due to this computational intractability, people resort to *non-exact* algorithms for solving large COP instances. These non-exact algorithms sacrifice the guar-

[1]This abbreviation can also stand for Constraint Optimization Problem. In this thesis, we associate the term COP with Combinatorial Optimization Problem, where the solutions are discrete.

antee of finding an optimal solution in order to find 'good enough' solutions in a *reasonable* computation time.

In the last few decades, a form of such non-exact algorithms called Stochastic Local Search (SLS) algorithms emerged as potential tools for solving COPs. Basically, SLS algorithms search the potential combinatorial solutions by *iteratively* modifying *parts* of the solution. SLS algorithms do not guarantee the optimality of the solutions but they turn out to be effective in practice.

The Meta-Level SLS Design and Tuning Problem **(DTP) (Section 3.1 - 3.4)**

Creating a simple SLS algorithm for a given COP is often easy. All one needs to do is to instantiate certain SLS components, set some parameters with (usually) default values, and run the algorithm on a set of COP instances. This process has been further simplified by using software frameworks or class libraries[2].

However, to engineer the SLS algorithm to perform *well* is *difficult*. An SLS algorithm is a success if its implementation can obtain *acceptable* quality solutions over a given set of COP instances and do so in *a reasonable amount of time*. Only in this sense an SLS algorithm is said to edge out exact algorithms for a given COP in practical cases.

The challenge: The performance of SLS algorithms for solving a given $\mathcal{NP}$-complete COP depends on many usually correlated factors. Different COPs or even instances of the same COP often require customized and holistic configuration of the SLS algorithm (set of search parameters, components, and search strategies) so that it performs well. Otherwise, the SLS algorithm performance is just 'average' or even 'poor'. Most of the time, we cannot just use the off-the-shelf SLS algorithm as its initial performance is so poor and must be redesigned to better suit the COP at hand. This is what we call the SLS Design and Tuning Problem (DTP).

It is not easy to address the SLS DTP as there are many things about the SLS algorithms or the COPs that are not well understood yet. Some algorithm designers, influenced by their past knowledge and experiences, resort to a trial-and-error approach through some 'random' experiments – which is tedious, or through well designed experiments (better but still laborious). Others created 'parameterless' SLS algorithms, in which the algorithms will self-adjust during the search. While this self-correcting (a.k.a reactive) strategy is interesting, it is not trivial to come up with a good reactive strategy every time. From an engineering perspective, many of these traditional approaches are not productive especially against a backdrop of tight development schedules.

The SLS DTP is naturally faced by *every* algorithm designer whenever he develops an SLS algorithm for a given COP. Typically, tackling the SLS DTP constitutes the bulk of SLS algorithm development time. Thus, if we have a better way to address this important issue, we can save a lot of time in building a sophisticated SLS algorithm implementation given a COP.

[2]As a note, we have built one such SLS software framework in the past: MDF [85, 84].

Addressing the SLS Design and Tuning Problem (Section 3.5.1 - 3.5.3)

To date, there are several researchers working on various parts of this SLS DTP issue. We classify their works into two parts: black-box or white-box approaches.

Black-box approaches are tuning algorithms for automatically configuring SLS algorithms. These tuning algorithms explore the configuration space and return the best found configuration within limited tuning time. This is an obvious improvement over manual trial-and-error as the computer is the one doing the tedious work. These tuning algorithms typically employ some experimental design techniques such that potentially promising configurations are tried more thoroughly.

On the other hand, white-box approaches are methodologies, techniques, or tools that are created to assist algorithm designers in designing better SLS algorithms by opening up the 'box' (examine the details of SLS algorithm behavior). This offers insights on the inner working of the SLS algorithm which can inspire the algorithm designers in coming up with good search strategies, in choosing appropriate SLS components, or in reducing the potential domain sizes of the parameter values.

Previously, no single approach is clearly superior to address *all* types and aspects of SLS DTP. Black-box approaches are simple to apply, but will not help if the best configuration is 'outside the box' of the initial configuration space. White-box approaches can empower the algorithm designer to get insights on the search process, which is necessary for designing a better search strategy, but are less effective for fine-tuning.

1.2 Our Contributions

To tackle this SLS DTP, we propose a collaboration strategy between the human[3] and the computer. This collaboration is possible because the human (e.g. with his intelligence, visual perception abilities, common sense, etc) and the computer (e.g. with its speed, consistency, endurance, etc) complement each other. This promising collaboration is used in two ways. First, we utilize information visualization as the bridging interface between the human and the computer to form a novel white-box SLS *visualization*. Second, the individual strengths of human and computer in white-box and black-box approaches, respectively, are combined in an *integrated approach*.

Fitness Landscape Search Trajectory (FLST) Visualization (Chapter 4)

Our novel visualization interface for SLS white-box analysis is a visualization of a COP fitness landscape and the SLS trajectories on it. It is known in the literature that the fitness landscape structure of a COP instance affects the behavior of an SLS algorithm that is searching over it. To design an effective SLS algorithm for a particular COP, a preliminary fitness landscape analysis of various instances of the COP should be done.

However, even the fitness landscape of a moderate-size COP instance is already too big (exponential size) to be exhaustively explored. Various analytical metrics currently

[3]The terms 'human', 'one', 'he', 'user', or 'algorithm designer' are interchangeable. We use the term 'he' rather than 'he/she' to refer to third person for simplicity.

available are inadequate to analyze the search trajectory details of SLS runs. To answer this challenge, we have invented a visualization technique called *Fitness Landscape Search Trajectory* (FSLT) visualization, explained in detail in Chapter 4 and presented in our papers [83, 56, 59, 60, 57, 61, 58].

The SLS Visualization Tool: Viz (Chapter 5)

We have built an SLS visualization tool called Viz which implements these FLST visualization techniques and a few other existing white-box visualizations. Viz has several information visualization and user interface techniques which is helpful for analyzing SLS algorithm behavior. This tool Viz is presented in Chapter 5. Papers about Viz can be found in [59, 60].

The Integrated White+Black Box Approach (Chapter 6)

However, a white-box approach in the form of visualization alone is not suitable for fine-tuning the SLS parameters (which is a natural task for a black-box approach). Thus, we develop the Integrated White+Black Box Approach (IWBBA). We first start with a working SLS algorithm for a given COP. Then, we seek to understand the SLS algorithm behavior on the fitness landscape of various COP instances using the white-box FLST visualization mentioned previously. Then, we use the knowledge to (re)design the SLS algorithm (add potential search strategies, choose appropriate SLS components, or set good parameter values ranges). The tweaked SLS algorithm is then fine-tuned using a black-box tuning algorithm.

IWBBA utilizes the strengths of both white-box and black-box approaches to produce better results than either alone. To support IWBBA, we have extended Viz into an SLS engineering tool by adding the black-box component. This integrated approach using Viz is elaborated in detail in Chapter 6 and has been published in [61].

The Experimental Results (Chapter 7)

We applied our integrated approach using our Viz tool to design and tune various SLS algorithm implementations for several COPs. We managed to identify various fitness landscape characteristics, e.g. 'big valley', 'spread-smooth', 'spread-rugged', and 'golf-course-like' and have designed custom strategies for each of these fitness landscapes. These strategies work better than the original or baseline SLS algorithms, most notably the state-of-the-art Tabu Search for solving the Low Autocorrelation Binary Sequence (LABS) problem [58]. Details of our experimental results are shown in Chapter 7.

In one paragraph, this PhD thesis can be summarized as follows:

> Integrated White+Black Box Approach, a collaboration between the human (algorithm designer) to design effective search strategies via Fitness Landscape Search Trajectory visualization (white-box) and the machine to do fine-tuning using a tuning algorithm (black-box), is an effective

methodology to address the SLS Design and Tuning Problem – a meta-level problem of finding a good performing SLS algorithm given a COP.

1.3 The Structure of this Thesis

Main Body

The main body of this thesis is organized as follows:

1. This introductory chapter.
2. In Chapter 2, we elaborate the background material: SLS algorithms as effective way to solve COPs and the challenges that it faces.
3. In Chapter 3, we present the main problem: improving the performance of SLS algorithms on various COPs and situations – the SLS Design and Tuning Problem (DTP).
4. In Chapter 4, we explain our novel and intuitive visualization for analyzing the SLS trajectory behavior on a COP fitness landscape: the Fitness Landscape Search Trajectory (FLST) visualization.
5. In Chapter 5, we describe the other SLS visualizations and the user interface aspects of our SLS visualization tool: Viz.
6. In Chapter 6, we present our overall approach for addressing the SLS DTP:
   1. Allow the algorithm designer to examine, explain SLS algorithm behavior, and tweak it (the human and white-box part)
   2. Do fine-tuning using tuning algorithms (the machine and black-box part).
   We call this: the Integrated White+Black Box Approach (IWBBA).
7. In Chapter 7, we apply IWBBA using Viz on several SLS DTP scenarios.
8. In Chapter 8, we conclude our thesis by restating the contributions and discuss future work.

Appendices

The supporting materials are organized as appendices:

A. Details of COPs used in this thesis.

B. Details of SLS algorithms M and their configurations Φ used in this thesis.

C. Short discussion of Human Computer Interaction (HCI).

1.4 List of Publications

Parts of the material presented in this PhD Thesis are based on the following work that was carried out by the author together with his supervisors and colleagues during his PhD candidature at School of Computing, National University of Singapore, from August 2004 to July 2009:

Chapter in Book:

1. **S. Halim** and H.C. Lau. Tuning Tabu Search Strategies via Visual Diagnosis. In *Metaheuristics: Progress in Complex Systems Optimization*, ch 19, pp 365-388, 2007. Springer. [56].

Conference/Workshop Full Papers:

1. **S. Halim**, R.H.C. Yap, and H.C. Lau. An Integrated White+Black Box Approach for Designing and Tuning Stochastic Local Search. In *Principles and Practice of Constraint Programming (CP)*, pp 332-347, 2007. [61].
2. **S. Halim** and R.H.C. Yap. Designing and Tuning SLS through Animation and Graphics - an extended walk-through. In *Engineering Stochastic Local Search Workshop (SLS)*, pp 16-30, 2007. [57]. Cited by: [86].
3. **S. Halim**, R.H.C. Yap and H.C. Lau. Viz: A Visual Analysis Suite for Explaining Local Search Behavior. In *User Interface Software and Technology (UIST)*, pp 56-66, 2006. [60]. Cited by: [33, 142, 80].
4. H.C. Lau, W.C. Wan, and **S. Halim**. Tuning Tabu Search Strategies via Visual Diagnosis. In *Metaheuristics International Conference (MIC)*, pp 630-636, 2005. [83]. Cited by: [109, 141].

Conference Poster Papers:

1. **S. Halim**, R.H.C. Yap, and F. Halim. Engineering Stochastic Local Search for the Low Autocorrelation Binary Sequence Problem. In *Principles and Practice of Constraint Programming (CP)*, pp 640-645, 2008. [58]. Cited by: [135, 148].
2. **S. Halim**, R.H.C. Yap, and H.C. Lau. Visualization for Analyzing Trajectory-based Metaheuristic Search Algorithms. In *European Conference on Artificial Intelligence (ECAI)*, pp 703-704, 2006. [59].

Oral Presentations at International Conferences/Workshops:

1. 1st Stochastic Local Search Workshop: September 6-8, 2007, Brussels, Belgium. Presenting [57].
2. 19th User Interface Software and Technology: October 15-18, 2006, Montreux, Switzerland. Presenting [60].
3. 6th Metaheuristics International Conference: August 22-26, 2005, Vienna, Austria. Presenting [83].

Chapter 2

Background

If I have seen a little further, it is by standing on the shoulders of Giants.
— **Isaac Newton**

This chapter provides background material for this thesis. We briefly discuss $\mathcal{NP}$-complete Combinatorial Optimization Problems (COPs), the trade-offs between exact and non-exact algorithms, and then extensively discuss various aspects of non-exact Stochastic Local Search (SLS) algorithms. Studying the material presented in this chapter is necessary to appreciate the rest of this PhD thesis, especially the parts about terms and notations which are often not standard across different resources in the literature.

2.1 $\mathcal{NP}$-complete Combinatorial Optimization Problems

> Grace owns a small food delivery service. Every day, orders come and she needs to plan a schedule to serve these orders using her only car. With approximately 30 customers per day scattered around the neighborhood, Grace faces about 30! possible routes to choose from. Each route requires a different amount of traveling distance/time and thus operating cost. To maximize profit, Grace must choose the shortest route which will save traveling distance. This problem can be modeled as a Traveling Salesman Problem (TSP), where the objective is to minimize the sum of the distance traveled by the car. By using a good solver for this problem, Grace can save operating cost and thus increase her company's profit.

The simplified example above is a single-objective Combinatorial Optimization Problem (COP). The solutions for this COP are discrete[1] – a set of car routes. COPs

[1]There are optimization problems where the solutions are continuous. Some continuous optimization problems that can be formulated as linear programs are easier to solve as one can use established linear programming techniques like SIMPLEX [154]. There are also multi-objective optimization problems where we need to optimize with respect to several, usually conflicting, objectives. Continuous and multi-objective optimization problems are outside the scope of this thesis.

like this are found in many other real-life applications, e.g. the application of Traveling Salesman Problem to real-life circuit drilling problem [73, 143], the application of Quadratic Assignment Problem to real-life hospital or keyboard layout problems [137, 114], the application of Low Autocorrelation Binary Sequence Problem to communication and electrical engineering [87, 58], and many others in the field of Engineering, Operations Research, Science, Computational Biology, etc. More complete references of the COPs used in this thesis are listed in Appendix A.

A *minimizing*[2] COP P consists of [89]:

- A set D_P of instances,
- A finite set $S(\pi)$ of candidate solutions for each instance $\pi \in D_P$, and
- An objective function g that assigns a rational number $g(s)$ called the Objective Value (OV) for s to each candidate solution $s \in S(\pi)$.

An optimal solution for an instance $\pi \in D_p$ is a candidate solution $s^* \in S(\pi)$ such that, for all $s \in S(\pi)$, $g(s^*) \leq g(s)$.

Considering that COPs have many practical uses and good solutions can bring profit and efficiency, it is important to have good solvers for these COPs. However, finding good solvers is a real challenge, despite the advances in algorithms.

The major source of the hardness[3] of these COPs is because many[4] of the COPs are $\mathcal{NP}$-complete [47].

The fact that many important COPs are $\mathcal{NP}$-complete, renders the naïve exhaustive enumeration approach impractical as the problem instance size increases. The hardness of these COPs is the motivation for research in optimization algorithms and numerous proposals have been published to address it.

2.2 Algorithms for Solving COPs

There are many algorithms for solving COPs. Basically they can be classified into two extreme paradigms: *exact* versus *non-exact* algorithms.

[2]Without loss of generality, every maximizing COP can be converted into a minimizing COP by multiplying the objective function by -1.

[3]The runtime in practice among the solvers of these $\mathcal{NP}$-complete COPs varies, e.g. finding good solutions for QAP is typically more difficult than for TSP for the same input size n. See Appendix A for details.

[4]There exist COPs that are not $\mathcal{NP}$-complete, for example: the Minimum Spanning Tree (MST) problem, where one needs to select V-1 out of E edges such that the graph is connected, has no cycle, and the total length of selected edges is minimum. For this MST problem, we also cannot use the naïve enumeration algorithms but there exist efficient greedy algorithms, such as: Kruskal or Prim algorithms [28], which are adequate for solving this problem in polynomial time. These non $\mathcal{NP}$-complete COPs are outside the scope of this thesis.

2.2.1 Exact Algorithms

Exact algorithms are clever version of the exhaustive enumeration approach. They are complete in the sense that the existence of feasible and then optimal solution(s) can be determined with certainty once an exact algorithm has successfully terminated.

Examples of exact algorithms are Branch and Bound (B&B) [154], Constraint Programming (CP) [10], etc[5]. In B&B, we prune the search sub-tree once we can guarantee that there is no solution under that sub-tree which will give a better result than the best currently known. With such pruning techniques, B&B effectively searches in a smaller search space. In CP, we use domain reduction, constraint propagation, and systematic search to find best feasible solutions.

While exact algorithms are guaranteed to deliver the optimal solution (if any), the drawback is their running time. Their runtimes render these algorithms impractical for reasonably big instance sizes that are commonly found in real-life settings.

There are research to further advance these exact algorithms: better speed, tighter bounds, exploiting symmetries, better cutting planes, etc. However, notwithstanding the importance of exact algorithms, it seems that exact algorithms are not the best approach for solving *large* COP instances in *reasonable* time.

2.2.2 Non-Exact Algorithms

For practical usage, especially in a real-time setting, faster techniques are needed. This is where the other paradigm, the non-exact algorithms, are used, e.g. heuristics, Stochastic Local Search. The reasonable assumption that a user will already be satisfied if he can consistently obtain good enough[6] solutions within reasonable time, led researchers to develop various non-exact algorithms with this idea: sacrifice the guarantee of feasibility and optimality (completeness) for (much) faster speed. But there are limitations: we are unable to state whether a feasible solution exists when a non-exact algorithm terminates before finding one. We are also unable to measure the quality gap of the solution produced by non-exact algorithms w.r.t the optimal solution.

Heuristics and Stochastic Local Search Algorithms

Heuristics are simple (usually greedy) techniques, which seek good solutions at a reasonable computational cost, e.g. in low-order polynomial time. Heuristic algorithms are usually derived from the characteristic of good solutions. For example: good TSP tours have short edges. Thus, we can start the tour from any city; then greedily pick the nearest neighbor one by one to complete the TSP tour. This typically produces a string of short edges except for the last few edges. This greedy 'nearest neighbor' heuristic has no performance guarantee but it can perform well on several TSP instances.

[5]There are other techniques like Branch and Cut (B&C), Integer Programming (IP), etc. Interested readers can browse books like [154].

[6]The term 'good enough' here refers to solutions which are close (if not equal) to the optimal solution, or within certain acceptable quality bound that already satisfy the user's needs.

The major weakness of greedy heuristics is that they are confined into small search spaces only, which may be severely sub-optimal, unless this heuristic is tailored to solve the given COP well[7]. A heuristic does not actually 'search' the COP search space, it just produces a specific solution based on its strategy. To be able to 'navigate' along the search space in the quest for better solutions, a more generic, meta-level controller is needed. This is where Stochastic Local Search (SLS) algorithm comes into the picture. We discuss these SLS algorithms in-depth in Section 2.3.

2.2.3 Comparison between Exact versus Non-Exact Algorithms

The trade-offs exhibited by the two paradigms above can be summarized as follows: "exact algorithms produce optimal solutions in a much longer time than non-exact algorithms" and "non-exact algorithms can give solutions with reasonable quality in much shorter time than exact algorithms".

The fact that exact algorithms are slow does not imply that they are useless. When optimality is a must or if the COP is so overly constrained that we want to find whether a solution exists, we do not have any choice other than using exact algorithms. Also, exact algorithms should be preferred when the instance size is *small enough* for exact algorithms to be run within the given running time.

But when the instance size is large, a well designed non-exact algorithm that is fast and produces reasonably good solutions is an attractive or even the only option.

Nowadays, the distinctions between the two paradigms are blurred as sometimes they are merged to form a stronger hybrid. For example, the bounding formula in B&B may be determined via heuristic-like methods, merging CP and SLS algorithm (e.g. [112]).

2.3 SLS Algorithms for Solving COPs

2.3.1 Background

The term 'Stochastic Local Search (SLS)' is made popular by [68]. The term 'SLS' is more general[8] than the similar term 'Metaheuristic (MH)'[9].

SLS algorithms have been evolving in recent years. Since the pioneering works of meta-controller to escape local optima in mid 1980s (e.g. Simulated Annealing [78], Steepest Ascent Mildest Descent [62], and Tabu Search [50]), there are now many more SLS algorithms. The list below highlights some of the popular ones:

[7]For example, a good heuristic for TSP is the Lin-Kernighan (LK) heuristic [73].

[8]There is a formal definition for SLS algorithms whereas this is not true for metaheuristics. Iterative improvement algorithms that are in some way randomized (e.g. by randomizing the order of searching neighborhood in first-improvement) belong to SLS algorithm, but these algorithms are usually not considered as metaheuristics. The term general-purpose SLS method is somehow equivalent to metaheuristic. For more details, see [68].

[9]This term is derived from Greek words: '$\mu\epsilon\tau\alpha$' (meta, which means: to guide; a higher level) and '$\epsilon\upsilon\rho\iota\sigma\kappa\epsilon\iota\nu$' (heuriskien - eureka, which means: to find or to discover).

1. Tabu Search (TS) [50, 52, 116]
2. Iterative Local Search (ILS) [133]
3. Simulated Annealing (SA) [78]
4. Genetic Algorithms (GA) [65]
5. Scatter Search (SS) [117]
6. Ant Colony Optimization (ACO) [37]
7. Variable Neighborhood Search (VNS) [63]
8. Memetic Algorithm (MA) [89], etc

Today, SLS algorithms are among the most efficient and robust optimization strategies to find high-quality solutions for many real-life COPs (see Section 2.1 and Appendix A). A large number of successful applications of SLS algorithms in various fields are reported in the literature: in books and journals [105, 2, 150, 19, 51, 106, 23, 68, 117, 39], as well as in scientific meetings, e.g. Metaheuristics International Conference (bi-annually since 1995) [107, 151, 121, 120, 71, 34, 29], Engineering SLS Workshop (bi-annually since 2007) [134, 130], CP, LION, META, CP-AI-OR, INFORMS, EU/ME, etc.

2.3.2 What is an SLS Algorithm?

An SLS algorithm can be described from various angles. We list some of them below:

1. SLS algorithm is an iterative non-exact algorithm. An SLS algorithm will not stop upon encountering the optimal solution during the search as it does not know that the solution is indeed optimal. It will only stop once it satisfies the user-adjustable termination criteria and return the current best found solution. Quality-wise, the solutions found by SLS algorithm are *usually* better compared to the solutions found by the more simpler non-exact algorithms (e.g. simple greedy heuristic) as the SLS algorithm searches the fitness landscape rather than just constructing one solution only.

2. SLS algorithm combines intelligent exploitation and exploration of the fitness landscape in order to find good solutions efficiently. Decisions of where to move are taken using local knowledge only, perhaps with some influence of randomness to achieve more robustness [52]. This part is elaborated in Section 2.3.3.

3. Anatomically, an SLS algorithm can be divided into two parts: an algorithmic template M and a configuration Φ. The behavior and performance of the SLS algorithm M are highly dependent on the chosen Φ: search parameters, components, and search strategies. A more detailed explanation is in Section 2.3.4.

Additional Definitions for COP and SLS Algorithms

Formally, an SLS algorithm for a COP can be defined as given below. These definitions assume a **minimizing** COP. They are adopted from various sources (e.g. [89, 68]) and enhanced with our own additional definitions:

Local Move: A small transformation from a current solution s into s', both s and $s' \in S(\pi)$, e.g. swapping two cities in a TSP tour.

Improving Move: A local move where the objective value $g(s') < g(s)$.

Non-Improving Move: A local move where the objective value $g(s') \geq g(s)$.

Neighborhood $N(s)$: The neighborhood operator (set of potential local moves) used by the SLS algorithm for traversing the search space $S(\pi)$.

Distance Function $d(s_1, s_2) : S \times S \rightarrow \Re$: This function calculates the differences between two solutions s_1, s_2 in an n-dimensional space $S(\pi)$. Various distance functions are discussed in [74, 123, 124, 45, 89, 68, 128].

Fitness Landscape FL: $\langle S(\pi), d(s_1, s_2), g(s) \rangle$. FL is the space where the SLS algorithm operates (see Section 2.3.3 for an illustration). FL is a wider definition than Search Space $S(\pi)$.

Region R: A region in FL is a set $R \subseteq S(\pi)$ such that for each pair of solutions $s', s'' \in R$, s' and s'' are connected, that is, there exists a connecting path $s' = s_0, s_1, \ldots, s_k = s''$, $\forall s_i \in R$ and $s_{i+1} \in N(s_i)$ $\forall i \in \{0, 1, \ldots k-1\}$.

Infeasible Region IR: An infeasible region IR is a region where all the solutions of IR are infeasible, that is, the solutions do not satisfy the constraints of the COP.

Search Trajectory ST: ST starts with an initial solution s^0. At iteration $t > 0$, ST moves from solution s^{t-1} to s^t and so on until the SLS algorithm stops, i.e. $ST = s^0 \rightarrow s^1 \rightarrow \ldots \rightarrow s^{lastItr}$. $|ST|$ denotes the number of solutions along ST. In population-based search, this term becomes plural: search trajectories.

Local Optima/Minima LO: A solution s is a Local Optima LO if $g(LO) \leq g(s'), \forall s' \in N(LO)$. The notion of local optimality is only w.r.t neighborhood N. If we change N, the set of LO will be likely different. $|LO|$ denotes the number of local optima for a particular COP instance.

Global Optima/Minima GO: A solution s is a Global Optima GO (optimal solution) if $g(GO) \leq g(s'), \forall s' \in S(\pi)$. $|GO|$ denotes the number of global optima for a particular COP instance.

Best Found BF: A solution is the best found solution if $g(BF) \leq g(s'), \forall s' \in ST$. BF may or may not be equal to GO.

Best Known BK: The best known solution for a COP instance, found by exact algorithm (guaranteed optimal) or the best ever solution found by any SLS algorithm (BK may or may not be GO).

2.3.3 Walks on a COP Fitness Landscape

The SLS algorithm behavior can be abstractly described using an illustrative example of a 2-D *minimizing* COP as shown in Figure 2.1. The search space $S(\pi)$ of this COP instance π are all possible instantiations of the COP decision variables (x and y-axis). The set of all solutions $S(\pi)$, their pairwise distance from each other $d(s_1, s_2)$, and their corresponding objective values $g(s)$ (z-axis) form the *fitness landscape* FL. In this illustration, we discretize the x and y-axis values in increments of 0.1. This is illustrated by drawing the FL using *wireframe*. The letters A, B, C, D, E, F, G, H, I, and P denote some solutions in the fitness landscape, each with its own objective

value (the lower the better). The solid **red line** illustrates the *search trajectory* ST – a series of solutions traversed during the search. The objective of ST is to hit the goal (e.g. I: the global optimum GO).

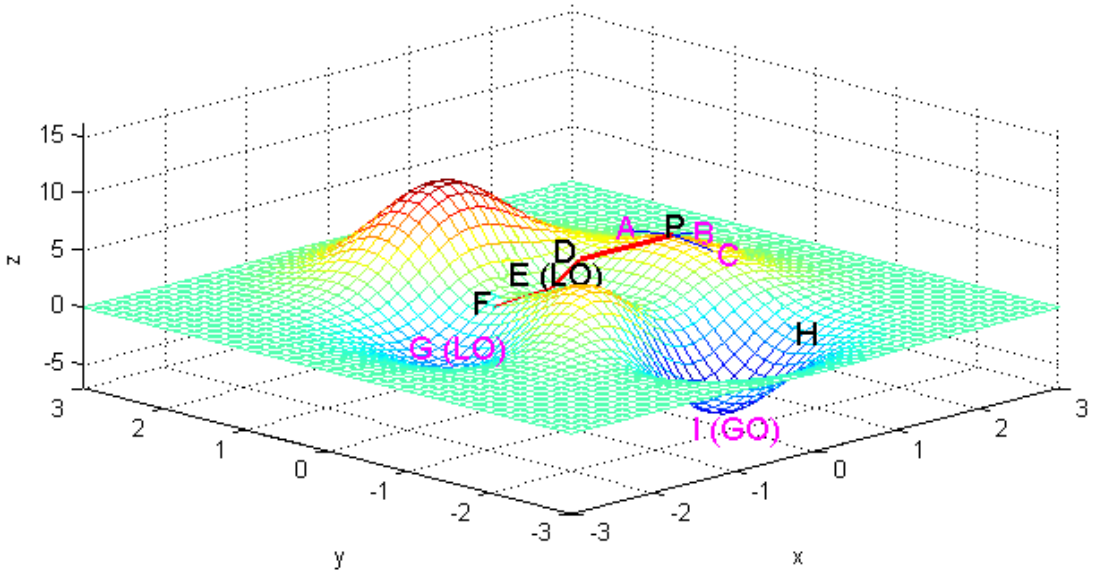


Figure 2.1: A Walk on a Fitness Landscape [126]: P-current solution; {**A**, **B**, **C**, D}-neighbors of P; {E, **G**}-*LO*; {F, H}-another solutions; {**I**}-*GO*. See text for details.

Suppose that the current solution is P. At this point of time, the SLS algorithm determines a subset of solutions in the fitness landscape. In this case, the local neighborhood of $N(P)$ is currently $\{A, B, C, D\}$. Suppose the SLS algorithm's greedy heuristic selects a solution from $N(P)$ based on its *Objective Value* OV (e.g. improving move to D), then subsequently to E and F – an intensification. While the objective function provides guidance to steer the search, it may lead the SLS algorithm to be trapped in a local optimum LO, in this case the **red line** is approaching a local optimum G instead of the GO I.

SLS algorithm may use some form of *randomization* to help diversification. The SLS algorithm can perform a 'jump' to another solution outside the neighborhood at a random iteration. For example in Figure 2.1, the SLS algorithm can 'jump' from position F to H where distance $d(F, H) > \epsilon$ – a diversification. We can see that H happens to be closer to GO I and continuing the SLS algorithm from H may be useful. However, this *stochastic* component also implies that ST may vary in different runs, making SLS algorithm behavior analysis difficult[10].

[10]It is hard to predict SLS algorithm behavior. Given a complete SLS code and a COP instance π, one cannot clearly describe how the SLS algorithm will behave until it runs. If the SLS algorithm is not implemented properly, it may exhibits behavior well beyond the original intent of its design. For example, if an SLS algorithm is supposed to do an intensification but in fact it travels far from the original position in FL, its intensification strategy is **not** successful, regardless of the result that it managed to obtain. Even if the result is 'good', it is probably not due to the influence of the 'incorrect' strategy but may be due to another reason(s). This has been coined as the 'failure modes' in [153].

2.3.4 Algorithmic Template M + Configuration Φ

An SLS algorithm can also be viewed as an algorithm with two major parts [17]:

1. The **algorithmic template** M:

```
BASIC-SLS(InitS)
1  CurS ← BF ← InitS
2  while TerminationCriteria are-not-satisfied
3      do CurS ← Choose(NeighborhoodConstraints(CurS, parameters))
4         Update Data Structures
5         if Better(CurS, BF)
6           then BF ← CurS
7         SearchStrategies
8  return BF
```

2. Configurable parts that makes up the **configuration** Φ:

- **ParameterValues** (**bold** in algorithmic template M)
  Parameter values are adjustable numeric values within M which affect the overall SLS algorithm performance. The SLS algorithm performance is usually sensitive towards these parameter values. Usually, SLS algorithm experts can roughly gauge good ranges of these values via pilot experiments. Parameter values are usually not constant but often depend on instance characteristics (e.g. instance size). These parameter values must be set a priori and usually subject to a fine-tuning process. Several parameters are often correlated.

- Components (underlined in algorithmic template M)
  SLS algorithm components are essential parts of M that need to be chosen or implemented to make M work. Usually this is the place where the user places his sub-ordinate heuristics. Components can have some embedded parameter(s). Typically, SLS algorithm components are problem-specific, thus less sensitive across different instances. The choices of components are also often correlated [24]. With parameters and components properly set, the SLS algorithm is ready to be executed and produce solutions.

- SearchStrategies (small caps in algorithmic template M)
  Search strategies are optional part on top of M + {parameters + components} that are used to alter the behavior (trajectory) of the search, in bid to further improve the SLS algorithm performance. Without these search strategies, the SLS algorithm will still work. However, state-of-the-art SLS algorithms are usually those which employ good search strategies.

The selected $M + \Phi$ will influence[11] the behavior and thus the performance of the SLS algorithm. Finding the correct $M + \Phi$ combination is hard as each of these configurable parts can assume one of its valid domain values. The configuration set (space) that

[11] SLS methods with smaller configuration space is more preferred because it is easier to tune.

contains all possible instantiations of these configurable parts can thus be very large. This is the main issue in this thesis and it is discussed in Chapter 3.

The SLS algorithmic template M and configuration Φ of several well-known SLS algorithms used in this thesis are presented in Chapter 7 and in Appendix B.

2.3.5 Implementation Issues

Implementing an SLS algorithm is usually a straightforward job but there are different ways[12] to implement the same SLS algorithm, which may cause the resulting performance to be different.

Fortunately, the algorithmic template M of typical SLS algorithms is common from one application to another and some SLS components are also frequently reused. It is thus beneficial to apply software engineering principles by creating SLS software frameworks, systems, or libraries. This way, subsequent SLS algorithm implementations can be faster because one can reuse one's previous codes. There are several SLS software frameworks available in the literature, e.g. OpenTS [64], EASYLOCAL++ [32], HOTFRAME [44], LOCALIZER++ [91], COMET [149], and including our Metaheuristics Development Framework (MDF) [85, 84].

2.3.6 Performance Evaluation

SLS is Difficult to be Analyzed Theoretically

It is hard to show theoretically the effectiveness of the SLS algorithm. Usually we do not have much theoretical understanding for a particular SLS algorithm. And when we have such an understanding, the analysis is mainly worst-case and not average-case. Thus the information gained is not practical as it is only applicable under very restrictive assumptions, e.g. the proof of convergence for Simulated Annealing (SA) [98, 103] – it is said that SA will converge in the limit.

While discussing theoretical results of optimization algorithm, we need to mention the 'No Free Lunch' (NFL) theorem [155]. This theorem suggest that on average, no single SLS algorithm (TS, ILS, SA, ACO, GA, etc) is better than random search on *all* COPs. Success comes from adapting the SLS algorithm to the COP at hand. To have a good solver for a given class of COPs, one should produce specialized algorithms where each algorithm tackles a specific (subset) of instances of the problem, exploiting problem-specific information as much as possible while avoiding over-fitting (good for training instances but not on test instances).

Evaluating SLS Algorithm Performance via Empirical Analysis

Unlike exact methods for a given COP, which mainly compete with other exact methods in term of speed, the decision of whether a particular heuristic based strategy performs

[12] Examples: different data structure, algorithm details, code optimization, programming language, compiler, or computing platforms.

well or not is more difficult. We may need to devise a number of performance evaluation metrics, e.g. solution quality, consistency, etc (details in Section 3.5.2).

As mentioned before, an SLS algorithm is very hard to be analyzed theoretically. Currently, SLS algorithms are analyzed empirically, either via statistical techniques or via visualization. Such empirical analysis must be done properly, as highlighted in the literature [13, 66, 115, 40, 42, 49]. See the following example for illustration.

Suppose we run two SLS algorithms A and B on 4 different instances of a minimizing COP and we obtained these set of best found objective values:

A: {30, 20, 20, 20}, $mean(A) = 22.5$, $median(A) = 20$, $std\text{-}deviation(A) = 5$
B: {23, 23, 23, 23}, $mean(B) = 23$, $median(B) = 23$, $std\text{-}deviation(B) = 0$

Issue 1 - choice of descriptive statistics:
If we only report the mean values only, then we may favor SLS algorithm A over B but if we value robustness, then B is actually better since it is much more consistent and not too much different from the mean of A. We can also measure the gap between the mean and the median to check robustness.

Issue 2 - influence of 'outliers':
Now if we omit the first instance, then the 'apparent' performance will be as follows:
A: {20, 20, 20}, $mean(A) = median(A) = 20$, $std\text{-}deviation(A) = 0$
B: {23, 23, 23}, $mean(B) = median(B) = 23$, $std\text{-}deviation(B) = 0$
Selection of test instances plays a role in examining or comparing SLS algorithms!

Issue 3 - are the experiments significant? - inferential statistics:
After we run SLS algorithms A and B on a sample of 4 COP instances, we are interested to know whether our results can be generalized to a population of COP instances. This is the job of inferential statistics and this part is often omitted in SLS literature. For details about statistical techniques, refer to [55].

2.4 Summary

1. Combinatorial Optimization Problems (COPs) are important as there are many important real-life problems that can be modeled as COPs. However, solving COPs is difficult because they are usually $\mathcal{NP}$-complete.

2. There are two approaches for solving $\mathcal{NP}$-complete COPs: exact and non-exact algorithms, each with their own pros and cons. In this thesis, we focus on a family of non-exact algorithms falling under the Stochastic Local Search (SLS) definition.

3. We have presented various aspects of SLS algorithms: their historical background; SLS and COP terminologies, SLS algorithm as walks on COP fitness landscapes, SLS algorithm as consisting of two components: $M + \Phi$; SLS algorithm implementation and performance evaluation.

2.5 Looking Ahead

The performance of SLS algorithms strongly depends on the chosen $M + \Phi$. To obtain good performance, users need to find the correct configuration Φ for their chosen SLS algorithm M. There is 'no free lunch'!

This issue gives rise to the main problem addressed in this thesis: the SLS Design and Tuning Problem. We discuss it in the next chapter.

Chapter 3

The Stochastic Local Search Design and Tuning Problem

We keep moving forward, opening new doors, and doing new things, because we are curious and curiosity keeps leading us down new paths.
— **Walt Disney**

In this chapter, we discuss the SLS Design and Tuning Problem. *Parts of this chapter have been published in [83, 56].*

3.1 The Quest for Better Performance

It is obvious that one wants to engineer a better SLS algorithm if its current performance is low or when it varies greatly across different test instances beyond an acceptable level. However, we have seen that even if the current performance is already good, people are still looking for an even better performing SLS algorithm.

This natural quest for a better SLS algorithm performance is the main problem discussed in this thesis. We coin it as the SLS Design and Tuning Problem (DTP)[1].

The benefits of having a better SLS algorithm performance on a given COP is obvious, e.g. more efficiency, more time savings, more cost savings, more profits, etc.

Although the crucial importance of good algorithm design and proper tuning are acknowledged in the literature for a long time, *specific* works that are related to the tools and techniques for addressing the SLS DTP have emerged only *recently*, as shown in their time line below. This shows that more researchers are realizing the importance of dealing with the SLS DTP.

[1]Tuning is not only relevant for SLS algorithms, but also for either exact (e.g. [94]) or non-exact algorithms which have some 'tune-able' parts. Usually, modifying these 'tune-able' parts can yield different performance. We only discuss the **SLS** Design and Tuning Problem in this thesis.

- CALIBRA - Adenso-Diaz and Laguna **(2001-2006)** [3]
- F-Race - Birattari **(2002-2007)** [18, 17, 12], Yuan and Gallagher **(2005)** [157]
- Agent configuration algorithm - Monett-Diaz **(2004)** [99, 100]
- ParamILS - Hutter *et al.* **(2006-2008)** [69, 70]
- GGA - Ansótegui *et al.* **(2009)** [9]
- Tuning Wizard in ILOG OPL **(2008)** [72]
- Our works **(2004-2009)** [83, 56, 59, 60, 57, 61, 58]

3.2 Formal Definition of the SLS DTP

Let:

M**:** The basic algorithmic template of an SLS algorithm

Φ**:** The configuration of an SLS algorithm: (parameter values, components, and search strategies). See Section 2.3.4 for more details about $M + \Phi$.

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

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

T_{dev}**:** The development time to engineer the SLS algorithm: designing, implementing, analyzing, and tuning the SLS algorithm. T_{dev} for a real COP is usually limited[3].

T_{run}**:** The running or computation time[4] for the SLS algorithm to solve the given set of training instances I_{train} and/or test instances I_{test}.

Then, the SLS Design and Tuning Problem, abbreviated as **SLS DTP**, is defined as a multi-objective, multi-constraint problem:

> Find an SLS algorithm $M + \Phi$ within the development time T_{dev} that:
>
> 1. can obtain high quality (acceptable) solutions,
> 2. is robust, and
> 3. has fast run time (within T_{run}),
>
> when trained using instances from the set I_{train} and tested using other instances from the set I_{test} of the COP being solved.

By a common assumption, we *expect* the unknown future instances that are going to be solved by the SLS algorithm $(M + \Phi)$ to have characteristics similar to the instances I_{train} and I_{test}. Thus, the performance of the selected SLS algorithm $(M + \Phi)$ on these future instances is *expected* to be similar.

From the definition above, we can view the SLS DTP as another 'search problem', but in the configuration space. Since it is not well understood how to pick the best

[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 Section 2.3.6).

[3] We remark that the experiments in Chapter 7 of this thesis are done under no time-constraints.

[4] T_{run} is usually small in real-time applications, e.g. if a solution must be available every 1 hour, we cannot afford to have SLS algorithm that can give us the best solution but in more than 1 hour.

$M+\Phi$, addressing the SLS DTP is a process to learn the relationship[5] of $M+\Phi_1$, $M+\Phi_2$, ..., $M+\Phi_n$ w.r.t the performance of the SLS algorithm on various COP instances. 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 the SLS DTP as the size of configuration space is big. We believe that 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 [17] where the aspects of configuration Φ are now widened to include 'search strategies'. This may entail redesigning the SLS algorithm. In Table 3.1, we propose a classification for the SLS DTP that puts into perspective this view of 'adding search strategies'.

3.3 Classification of the SLS DTP

In the literature, the term 'tuning' is often too broad as it can mean "any action that makes an SLS algorithm perform better". To be more precise, we classify the SLS DTP into three types (see Table 3.1), according to the part that is being modified (compare with Section 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 the SLS DTP

Type-1: Calibrating Parameter Values

> Examples: calibrating tabu tenure (TS); perturbation strength (ILS); temperature T (SA); α, β, q_0, number of ants M (ACO); population size, recombination/mutation probability (GA); etc. The meaning of the term 'tabu tenure' is discussed below. For the other terms, please consult [51].

This is perhaps the 'easiest' type of SLS DTP: the SLS algorithm (SLS template M and SLS components) has been defined and all the algorithm designer needs to do is to set appropriate *parameter values*. For example, *tabu tenure* is a parameter of Tabu Search (TS) algorithm that controls how long (usually in terms of number of iterations) a certain local move is forbidden to be re-applied again. Tabu tenure parameter is usually one of the most influential parameter in TS algorithm.

Pellegrini *et al.* [109] show that different parameter values may influence the overall SLS algorithm performance. The challenge is that varying the value of one parameter may affect the optimal setting of the other parameter values since they are often correlated. Furthermore, in many practical situations, the range of parameter values is too large for the algorithm designer to determine the best values through trial-and-error.

[5] We remark that the definition of SLS DTP allows us to learn the relationship between $M_1 + \Phi_1$, $M_2 + \Phi_2$, etc (between different SLS algorithms), but this is a more difficult problem. In Chapter 7 of this thesis, we focus on designing and tuning SLS algorithms where the algorithmic template M does not change (i.e. given one SLS algorithm template M, find the best configuration Φ for it).

Type-2: Choosing Best Components

> Examples: choosing the best: local neighborhood (TS/SA/LS); tabu mechanism (TS); perturbation and acceptance criteria (ILS); cooling function (SA); pheromone table (ACO); recombination and mutation operator (GA); etc. The meaning of the term 'tabu mechanism' is discussed below. For the other terms, please consult [51].

In this type of SLS DTP, the algorithm designer needs to choose several components that will be used in a particular SLS algorithm. For example, *tabu mechanism* is a component of the TS algorithm that specifies how a certain local move is tabu. The algorithm designer can choose to record the list of solutions found within the last *tabu tenure* iterations or to record only certain attributes of those solutions in a data structure. The first approach completely prevents TS to revisit solutions found in the last *tabu tenure* iterations but slower. The second approach is faster but does not has such guarantee. The algorithm designer also has to choose the data structure to support this tabu mechanism, i.e. a linked list, array, or hash table.

Typically, each choice of SLS component has its own strengths and weaknesses. Charon and Hudry [24] 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 a substantial number of combinations. This type of SLS DTP is more complex than type-1, because choosing appropriate SLS components (this type-2) entails setting appropriate parameter values of the chosen components too (type-1).

Type-3: Adding (Dynamic) 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. The meaning of the term 'Reactive Tabu Search' is discussed below. For the other terms, please consult [51].

In this type of SLS DTP, the algorithm designer needs to design *additional* search strategies to improve the SLS algorithm *run-time dynamics*. Although the SLS algorithm already has some basic search strategy for exploring the fitness landscape, good additional search strategies may improve the overall performance in the short term: steer the search trajectory to more promising fitness landscape regions faster; and in the long term: do some diversification mechanism when the search stagnates, etc.

For example, *Reactive Tabu Search* [15] is a TS algorithm strategy where *tabu tenure* is adjusted based on the events encountered during the search. In overview, the strategy is as follows. When TS encounters many non improving moves, it reduces *tabu tenure* to encourage intensification. When TS encounters solution cycling, it increases *tabu tenure* to encourage diversification.

Unfortunately, search strategies are often problem-specific and deriving the correct ones is tricky as the number of possible search strategies can only be limited by one's own imagination. The effectiveness of search strategies may also depend on the chosen parameters of the strategy (type-1), e.g. how many non-improving iterations before 'restart strategy' are used, etc. If not set properly, the search may exhibit behavior well beyond the original intent of its design ('failure modes' in [153]), e.g. an *overuse* of 'random restart strategy' that moves SLS trajectory to an arbitrary point in search space can turn a complex SLS algorithm into a simple random walk heuristic.

This is perhaps the most difficult type of SLS DTP.

3.4 The Need for a Good Solution

Quotes from Various Researchers

A compilation of quotes from the experts in the field (full quotes in the footnotes) highlight both the practical importance and the difficulties of addressing the SLS DTP:

- Addressing the SLS DTP is itself a scientific endeavor.[6]
  – Barr *et al.* [13], Section 7.
- In the past (mid 1990s), addressing the SLS DTP was a pure art rather than science.[7]
  – Osman and Kelly (editors) [107], preface.
- We usually want the best performing SLS algorithm.[8]
  – Birattari [17], Chapter 1.
- It is easy to implement the quick-and-dirty SLS algorithms, but not the state-of-the-art ones.[9]
  – Birattari [17], Chapter 7.
- More development time is spent on fine-tuning than designing the SLS algorithm.[10]
  – Adenso-Diaz and Laguna [3], Section 1.

[6] "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."

[7] "Design of a good meta-heuristic remains an art. It depends on the skill and experience of the designer and the empirical computational experiments."

[8] "Aiming at the best is one of the most fundamental traits of intelligence. In all activities, human beings tend to maximize benefit or, equivalently, to minimize inconvenience in some context-dependent sense. 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/her as irrational."

[9] "Indeed, the most notable strength of [SLS algorithms] lies precisely in the fact that they are relatively easy to implement and that, therefore a quick-and-dirty version of an [SLS algorithm] for a given class of COPs to be solved can be produced by a practitioner in few days. Such quick-and-dirty implementations are typically capable of fair performance; nevertheless, when state-of-the-art results are desired, careful design choices [type-2] and an accurate tuning [type-1] are needed."

[10] "There is anecdotal evidence that about 10% of the time dedicated to designing and testing new (SLS algorithm) is spent on development, and the remaining 90% is consumed (by) fine-tuning (its) parameters."

- Optimizing SLS algorithm may require interaction of multiple components.[11]
  – Stützle [51], Chapter 11.
- Usually, adaptive/reactive/self tuning methods are preferred.[12]
  – Stützle [51], Chapter 11.
- The effective tools for addressing SLS DTP are needed.[13]
  – Hutter *et al.* [70], conclusion.

Handling the SLS Design and Tuning Problem in a Holistic Manner

Due to the difficulty of the SLS DTP, many algorithm designers chose to deal with the type-1 and type-2 problems only. This often results in average performance. One may have a good set of components of the SLS algorithm and have all its parameters properly tuned. But, if the SLS algorithm does not conduct an intelligent exploration of the fitness landscape, it will often be outperformed by a dynamic, adaptive, self-tuning, and more intelligent counterpart. We must look at the whole picture!

A simple example is Reactive-Tabu Search (Re-TS) [15]. Re-TS has a search strategy that adaptively adjusts the tabu tenure according to Re-TS performance. Re-TS will often outperform the original, Static-TS, on the set of *test* instances – even if the tabu tenure setting of Static-TS is the 'best' over the set of *training* instances.

Our classification in Table 3.1 puts this type-3 of SLS DTP into consideration. We believe that to obtain the best solution for the SLS DTP, all types of SLS DTP must be addressed properly, ideally in this order: start from type-2 (select the most appropriate SLS components), then solve type-1 (set the parameters of the chosen components). If problems arise concerning the performance, then tackle type-3 (add clever search strategies to navigate the search) and re-do type-1 again (set the parameters of the chosen components and strategies).

3.5 Literature Review

The most naïve way to address the SLS DTP is what we call as ad-hoc tuning. Ad-hoc here is that the algorithm designer does not use any tool to aid him in gaining insights but choose a configuration in an ad-hoc fashion, e.g. a trial-and-error process to find a better performing SLS algorithm.

A more systematic way is to utilize computing power rather than manual labor to deal with SLS DTP. When the algorithm designer is confronted with several potential $M + \Phi$ configurations to choose from, he can simply run them all and take the best result. But this is impractical when the configuration space is large.

[11] "Thus, one should keep in mind that the optimization of an Iterated Local Search may require more than optimization of the individual components"

[12] "... 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."

[13] "... the increasingly effective tools produced by this line of research will allow researchers to focus on the essential and scientifically valuable tasks in designing algorithms for solving hard problems ..."

In this section, we review more advanced approaches to address the SLS DTP that utilizes human and machine strengths in better ways. We group the approaches into two major types: black-box versus white-box approaches. The details of the classification are shown in Table 3.2.

Black-Box Approaches	**White-Box Approaches**
Definition and Human Role: 1. Automated tools to fine-tune the SLS algorithm in order to obtain a good configuration given initial configuration set. Treats SLS runs as 'black-box'. 2. Human role is to provide initial configuration set, wait, and possibly restart the tool using a different configuration set if the results are still unsatisfactory.	**Definition and Human Role:** 1. Open up the 'box' so the algorithm designers can check the inner workings of the SLS runs and to assist them in *designing* a better SLS algorithm. 2. Require direct collaboration with human. Analytical reports about SLS runs produced by white-box approaches must be analyzed using human intelligence.
Strengths: 1. Can relieve the burden of addressing mainly type-1 (SLS parameters) and also type-2 (SLS components) of the SLS DTP from the human.	**Strengths:** 1. More suitable to address type-3 (SLS strategies) and also type-2 (SLS components) of the SLS DTP. 2. Allows creativity and innovation. 3. Can obtain insights about the SLS algorithm or the COP.
Weaknesses: 1. Limits creativity and innovation. 2. Unsuitable for the SLS DTP type-3. 3. If the configuration space is too large, these black-box approaches can be slow. 4. Cannot be used to debug or improve the underlying design of the SLS algorithm. 5. We also do not learn anything about the SLS algorithm or the COP.	**Weaknesses:** 1. Human still needs to do the analysis. 2. Less suitable for the SLS DTP type-1. 3. Time required to deal with the SLS DTP depends on the user's expertise. 4. Results may be inconsistent. Two human users design and tune the SLS algorithm differently. 5. SLS algorithm behavior may be hard to understand.

Table 3.2: Details of Black-Box and White-Box Approaches

3.5.1 Black-Box Approaches

In this subsection, we discuss 5 black-box approaches: Meta SLS, CALIBRA, F-Race, ParamILS, and GGA.

Meta SLS

A better approach than trying all configurations is to use a Meta SLS to configure another SLS algorithm. For example: in Evolving ACO [20, 110], a Genetic Algorithm (GA) is used to tune the parameters of the underlying Ants Colony Optimization (ACO). These ACO parameters are embedded in the GA's chromosome. So, the GA

runs ACO several times to solve the underlying COP. The ACO results are used by the GA to determine the next population of ACO parameters. Similar examples called 'Meta-evolution' are mentioned in [11, 108].

Meta SLS techniques liberate the algorithm designer from setting parameter values manually, but the algorithm designer still has to come up with the Meta SLS by himself. The other black-box approaches below are the generic versions of Meta SLS that work with virtually any SLS algorithms that adhere with its interfaces[14].

CALIBRA – http://coruxa.epsig.uniovi.es/~adenso/file_d.html (2006)

Adenso-Diaz and Laguna [3] proposed a tool to automatically calibrate parameter values given pre-defined ranges or bounds for each parameter. It works by iteratively running the target SLS algorithm with various parameter values and then uses the solution quality feedback to determine which range of parameter values should be used in the next iteration. CALIBRA uses Taguchi's fractional factorial design and local search to iteratively narrow down the range of parameter values until they converge to a 'local minimum' of the configuration space. Otherwise, after maximum number of iterations has elapsed, CALIBRA will return the best set of parameters found so far.

CALIBRA has the following limitations. The current version (released in 2006) can only tune up to 5 parameters. The other parameters must be fixed to 'appropriate values'. Also, if the given parameter value ranges are too small, CALIBRA will be quickly trapped in a 'local minimum' of the configuration space.

F-Race – http://code.ulb.ac.be/iridia.activities.software.php (2005)

Birattari [18, 17] proposed a racing algorithm, a method that was previously known in the machine learning community, to address the SLS DTP. The racing algorithm (F-Race), paraphrased from his work, can be summarized as follows: First, feed the F-Race with a (possibly large) set of candidate configurations. F-Race will estimate the expected performance of the candidate configurations by empirically running the SLS algorithm with those configurations on the training instances one by one. The worst ones are discarded as soon as sufficient statistical evidence[15] is gathered against them. This allows a better allocation of computing power because rather than wasting time evaluating low-performance configurations, F-Race focuses on the assessment of the better ones. As a result, more data is gathered concerning the configurations that are deemed to yield 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 real life racing.

The number of possible configurations can be very large, thus by not trying every possible configuration blindly, F-Race is much better than a systematic brute force try-all approach. However, this 'combinatorial explosion' of the number of configurations

[14]Each black-box tool described in this section requires the configured SLS algorithm to interact with the tool (e.g. to read the SLS configuration and to report the result) via some 'inter-process communication'. This is not standard across various black-box tools.

[15]Birattari used Friedman non-parametric tests for this purpose, see [55].

is also the limitation of F-Race. As F-Race actually 'races' all candidate configurations before dropping them step by step, F-Race will require enormous computation time if the configuration size is (very) large which may possibly exceed the maximum allowed development time. To overcome this limitation, we can either: keep the size of initial configuration set to be relatively small, which is not an easy task, or use other tuning algorithms that try the candidate configurations one by one rather than race them.

This racing algorithm has been adopted in other research works, e.g. [156, 157, 16]. Some suggestions on the improvement strategies for F-Race are in [12].

ParamILS – http://www.cs.ubc.ca/labs/beta/Projects/ParamILS (2008)

Hutter *et al.* [70] proposed ParamILS, a black-box tuning algorithm that utilizes Iterated Local Search (ILS) to explore the parameter configuration space in order to find a parameter configuration that is good for the given training instances.

The main idea of ParamILS can be described as follows. Starting from a default configuration (either random or supplied by the user) as the incumbent solution, ParamILS will use the one-exchange neighborhood (modify one configurable part of the incumbent solution) in each search step using some heuristic. If the resulting configuration performs better, it will be accepted as a new incumbent solution, otherwise, it will be rejected and ParamILS will generate a new neighbor using the old incumbent solution. With some probability, ParamILS will set the incumbent solution to a random configuration. This process is repeated until the termination criteria are reached.

Unlike CALIBRA that limits the number of configurable parts to be 5, ParamILS does not have such a restriction.

The current available implementation of this ParamILS, which is still in beta stage, has been used by other researchers to optimize their SLS algorithms, e.g. [25].

GGA (2009)

Ansótegui *et al* [9] proposed GGA (Gender-based Genetic Algorithm), a robust, inherently parallel Genetic Algorithm to configure SLS algorithms automatically. In [9], the authors show several experimental case studies where GGA outperforms ParamILS in configuring SAT solvers.

3.5.2 White-Box Approaches

Algorithm designers have devised various techniques to assist them in understanding their SLS algorithms' behavior and performance. This information is essential in order to adjust their SLS algorithms correctly. These techniques are either in the form of statistical analysis or information visualization techniques. They are discussed below.

Descriptive Statistics

Descriptive statistics (e.g. central tendency, variability, etc) are used in several ways in SLS algorithm analysis.

The most widely used statistic is **'Solution Quality'** where the SLS algorithm performance is measured on benchmark instances (e.g. TSPLIB [143], QAPLIB [114], etc). We are interested to measure the quality gap between *Best Found* BF solution returned by the SLS algorithm w.r.t the *Best Known* BK solution (or an optimal solution found by an exact algorithm). This gap is usually measured via the following 'percentage-off' formula: $\frac{abs(OV-BK)}{BK} * 100.0\%$.

'Robustness' is another desirable statistic. Robustness analysis is used to measure the degree of variation of BF solutions found by different runs of the SLS algorithm. This is because the SLS algorithm may behaves non-deterministically resulting in different search trajectories on different runs.

Solution quality and robustness can be visualized as a graph of the Solution Quality Distribution, a.k.a Objective Value (OV) over time, like Figure 3.1.A: A 'robust and better' (**red solid lines** closer to BK OV and tighter) versus 'not robust and poorer' (**blue dashed lines** that are not too close to BK OV and more spread) SLS algorithm performance can be easily seen.

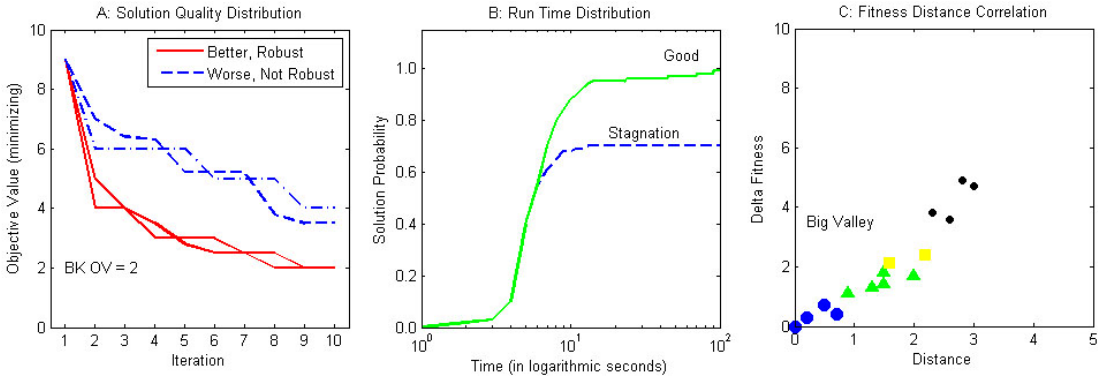


Figure 3.1: Example Visualizations of: A: Solution Quality Distribution (Robustness); B: Run Time Distribution; C: Fitness Distance Correlation

'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 BF solution. But, rather than using simple runtime statistics averaged over runs, a more realistic picture can be obtained by characterizing the **Run Time/Length Distribution (RTD/RLD)** where we measure the probability that the SLS algorithm manages to reach a certain target solution quality given a certain running time (the solution probability) [67, 129, 133]. In RTD visualization in Figure 3.1 (B), we observe that the **blue dashed line** exhibits stagnation behavior (the SLS algorithm is poor) as the solution probability does not increase even if we increase the running time exponentially (notice that the x-axis is in logarithmic scale) and the **green solid line** shows the expected exponential time behavior, i.e. the solution probability approaches 1.0 given much more time.

'Properties of the Fitness Landscape' of the COPs, such as the distribution and number of local optima, the existence of the 'big valleys', landscape ruggedness, the existence of plateau regions, etc, are known to affect the difficulty of a COP instance, which in turn influences the performance of the SLS algorithms. Although exploring the

entire fitness landscape is impractical, one may still gather crucial statistical properties of the fitness landscape by sampling the points in the fitness landscape.

Fitness Distance Correlation (FDC) analysis [74, 75, 45, 89, 68, 153] is used to measure the estimated difficulty of the fitness landscape. In FDC analysis, we sample a number of local optima. Let $F = \{f_i, f_2, \ldots, f_n\}$ be the fitness (objective) values of these n local optima with mean $\bar{f}$. Let $D = \{d_1, d_2, \ldots, d_n\}$ be the distance between each local optimum w.r.t. a reference point (usually the *nearest*[16] BF solution) with mean $\bar{d}$. The FDC coefficient, r_{FDC}, is defined as [74]:

$$r_{FDC} = \frac{cov_{FD}}{var_F * var_D} = \frac{\frac{1}{n-1} * \sum_{i=1}^{n}(f_i - \bar{f}) * (d_i - \bar{d})}{\frac{1}{n-1} * \sum_{i=1}^{n}(f_i - \bar{f})^2 * \frac{1}{n-1} * \sum_{i=1}^{n}(d_i - \bar{d})^2} \tag{3.1}$$

The result of such an analysis may yield interesting insights that can be exploited to improve the design of the SLS algorithms. High r_{FDC} coefficient (near 1.0) is a sign that there exists a correlation between fitness and distance. For example, the Traveling Salesman Problem (TSP) instances typically have high r_{FDC} (see an illustration of the FDC scatter plot in Figure 3.1 (C)). This implies that TSP solutions nearer to BF solution typically have good quality too, we can[17] design an SLS algorithm that concentrates on the region near the current BF solution.

Note that statistical methods like FDC analysis are not always accurate, e.g. a counter example for FDC analysis is in [6].

Inferential Statistics

While descriptive statistics about SLS runs give information about the *sample runs*, we also want to know if the observation is statistically valid. This information is valuable for making a judgement on how to improve the SLS performance. In this thesis, we use an inferential statistics technique called **Wilcoxon signed-ranks test** [55].

Wilcoxon signed-ranks test is a non-parametric statistical hypothesis test for repeated measurements on a single sample. This statistical test is used because the distribution of measurements of SLS runs is not normally distributed. With this test, we want to know whether the result of using one algorithm (e.g. the baseline algorithm) on the sample (e.g. the test instances) is significantly different (e.g. poorer) than the result of using another algorithm (e.g. the improved algorithm) on the same sample (e.g. the same test instances). The null hypothesis states that there is no difference. If the null hypothesis is true, then upon observing n different result pairs (the results of the baseline and improved algorithms), the Wilcoxon test statistic T (details on how to compute T is in [55]) is expected to be greater than the critical value V with confidence level α. But if $T \leq V$, we have to reject the null hypothesis and adopt the alternative hypothesis which says that there is a significant difference between the two results.

[16] As there exists a possibility that there are multiple BF, it is more appropriate to measure this FDC coefficient w.r.t the *nearest* one to avoid misleading the user by saying that the search is 'far' from the target, thus reporting low FDC coefficient, even though the search has actually managed to arrive at *another* solution which has the same OV as the BF.

[17] Fitness landscape analysis is heavily used in our thesis. More examples are shown in Chapter 7.

Statistical methods (both the descriptive and inferential statistics) can be used to help the algorithm designer to address all types of SLS DTP. However, this process is not straightforward as knowing the statistical information about the SLS algorithm or the COP is not sufficient to design a good SLS algorithm for that COP. A significant amount of human effort is still required to make use of the observations found using statistical analysis before a good solution for the SLS DTP can be produced. In the context of the SLS DTP, this lengthy process is undesirable due to tight development time. Fortunately, many computer-aided tools are available to gather and process the statistical data, e.g. EasyAnalyzer [33].

Human-Guided Search

Human-Guided Search (HuGS) utilizes human visual perception and intelligence by providing the user with a visualization and interaction tool (see Figure 3.2 for an example). HuGS presents a *problem-specific* visualization of the current solution (e.g. Vehicle Routing tours, etc) and allows the user to control the SLS algorithm (e.g. the user can focus the SLS algorithm to explore a subset of edges only, etc). This may work because the human knows the ingredients of good solutions of the COP and may be able to assist the SLS algorithm to obtain good results quicker.

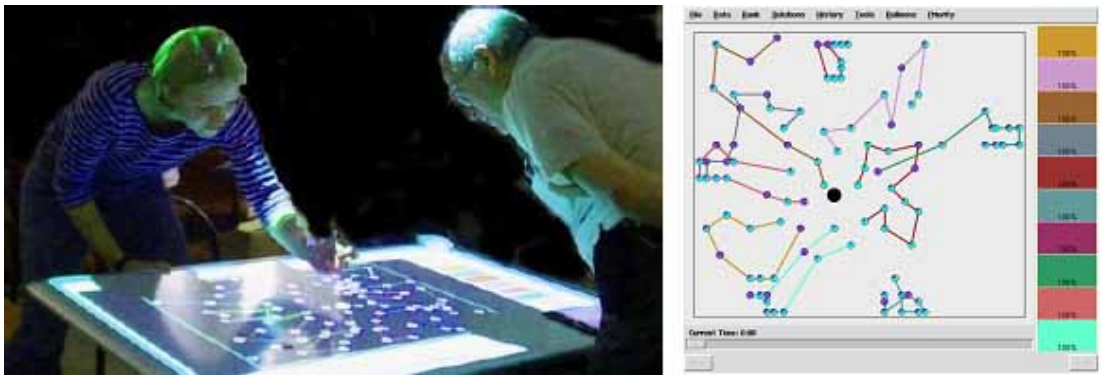


Figure 3.2: Human Guided Search (Figures are taken from [7])

Research on interactive man-machine optimization can be found as early as in the late 1960s [92, 81]. This line of work re-surfaced in Mitsubishi Electric Research Laboratory (MERL)'s projects in 2000s: [8, 7, 1, 79, 127, 26, 80].

This approach has a drawback. Guiding the SLS algorithm for a prolonged period is tedious and sometimes we do not know what to do to steer the search (see the explanation of Figure 5.8). Thus, the effectiveness of HuGS is limited by the stamina, patience, and intuition of the human user.

In its original intention, HuGS 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), and thus, HuGS can be classified as a white-box approach.

Visualization of Search Landscape and/or Behavior

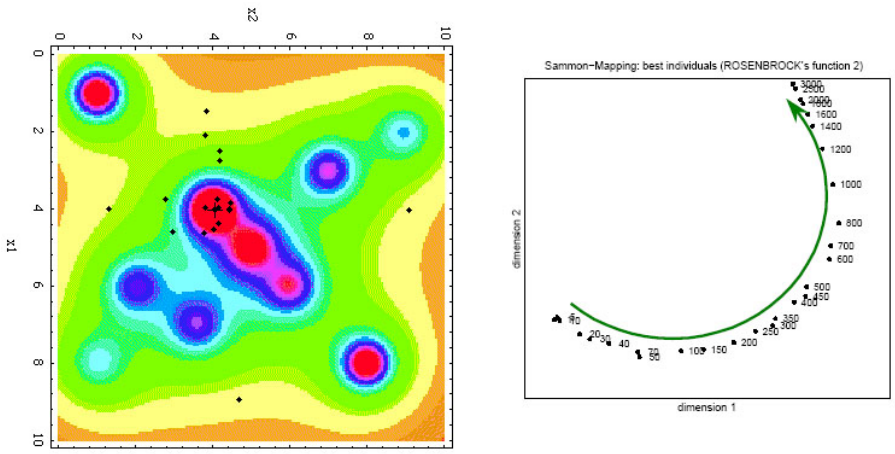


Figure 3.3: Left: Straightforward visualization of COP with only 2 decision variables by [136]; Right: Mapping 10-dimensional data set to 2-D in order to visualize "the path through the search space" by [111]. See text for details.

Syrjakow and Szczerbicka [136] proposed a visualization for analyzing Genetic Algorithms (GA) on a simple COP with 2 decision variables only (see Figure 3.3, left). Color is associated with the fitness of a region, with **red** being the good regions. **Black** dots describe the position of individuals in the GA population. The instance size is quite small so that the entire fitness landscape can be fully enumerated. The search positions are then animated in this visualization. This is intuitive, but this approach is limited to 'toy problems' only. In practice, $\mathcal{NP}$-Complete COP solutions are typically n-dimensional, which require some transformations to be displayed in a 2-D screen. Moreover, it is hard to enumerate the complete fitness landscape of an n-dimensional COP instance.

Pohlheim [111] proposed another visualization for analyzing evolutionary algorithms, e.g. GA. He picked the best individual of every generation in an evolutionary algorithm and then used a multidimensional scaling approach called Sammon-Mapping to transform the higher dimensional data set into 2-D data set. He then showed the 'path through the search space' induced by these best individual over generations. An example of visualizing the progress made by his algorithm on `ROSENBROCK's` test function [125] (10-dimensional) is shown in Figure 3.3, right. As it is hard to map high dimensional data set to 2-D, the author had chosen to keep the dimension and the number of points to be mapped to be very low (up to 10 dimension and about 30 best individuals from 30 generations in Figure 3.3, right).

Kadluczka *et al.* [76] proposed a visualization to analyze higher dimensions. The authors proposed a problem-specific mapping scheme for n-dimensional solutions to 2-D space which can be displayed on a 2-D screen. Then, 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 algorithm. This information can be used as a guide to improve the SLS algorithm.

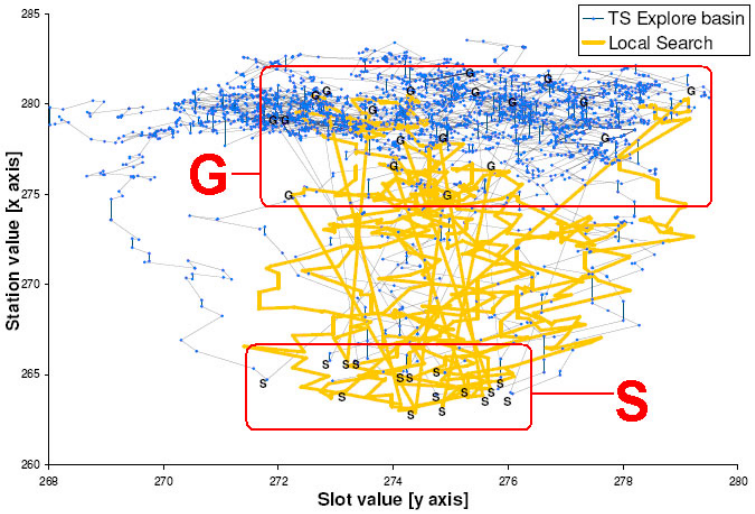
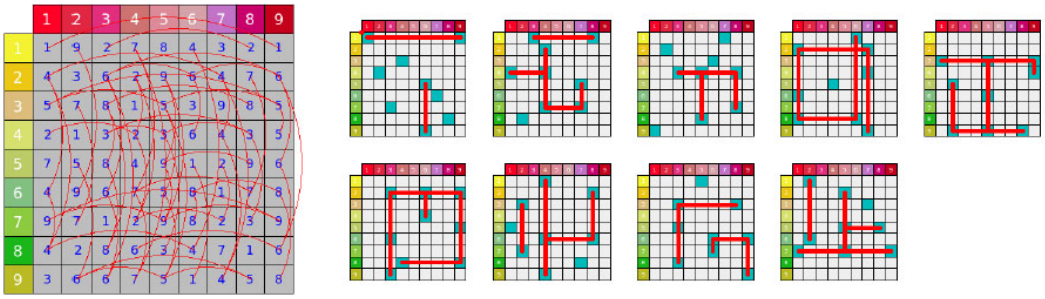


Figure 3.4: N-to-2-Space Mapping by [76]. See text for details.

In Figure 3.4, we observe that starting from the points denoted with S, we can trace the SLS algorithm attempts to reach various local optima solutions G. With such a visualization, we can gain insights of the SLS algorithm performance. We can see the search space covered by the SLS algorithm and conclude that Tabu Search (**blue**) explores the search space more thoroughly than simple Local Search (**orange**). The existence of several whitespace regions in the visualization implies that the regions are not yet explored, which implies that a diversification mechanism may be required.

The limitation of this visualization is that one must devise his own mapping scheme for each COP, with possibilities of cluttered visualization due to many n-dimensional solutions mapped to the same coordinate. Furthermore, the static visualization adopted in this work does not convey the SLS run-time dynamic behavior in a user friendly manner, e.g. it cannot show if the SLS algorithm is stuck in a local optimum (compare with Figure 4.13).



Sudoku Conflicts. Sudoku Conflicts (Variable/Value View).

Figure 3.5: Visualization of Search Behavior by [35]. See text for details.

Dooms *et al* [35] proposed a visualization of Constraint-Based Local Search (CBLS) written on top of their tool: COMET [149]. Figure 3.5 shows one example of their visualization on constraint conflicts in solving Sudoku puzzle. This visualization is animated, showing the changes of constraint conflicts over time as CBLS explores the solution space. However, with many lines changing over time as seen in Figure 3.5, it is quite hard to understand the runtime behavior of the underlying CBLS.

Mascia and Brunato [86] proposed a 3-D visualization to analyze the *search landscape* of a COP instance. This work is inspired by our earlier papers [57] and uses a similar spring-based method. We remark that 3-D visualization has more scalability issues as it needs much more data points than the corresponding 2-D visualization.

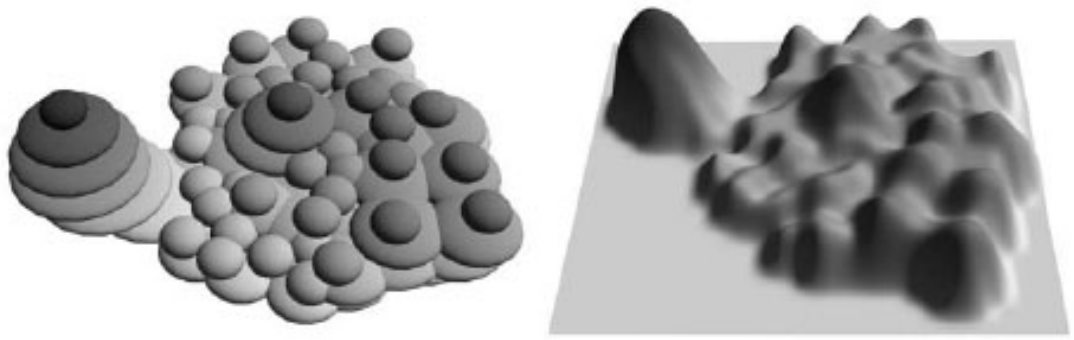


Figure 3.6: Visualization of Search Landscape by [86]. See text for details.

3.5.3 Comparison between Black-Box versus White-Box Approaches

In general, black-box approaches are suitable for fine-tuning parameter values. If the COP instances being solved are quite homogeneous, black-box approaches alone may be adequate to fine-tune the SLS algorithm to make it better. However, while black-box approaches can improve the SLS algorithm performance, they do so without explaining why it works. This is not ideal for advancing SLS algorithm research.

White-box approaches are more suitable for choosing appropriate SLS components and search strategies as determining these requires insights about both the COP and the SLS algorithm. When the COP being solved has heterogeneous instances, the algorithm designer may use white-box approaches to adapt the SLS algorithm to different COP instance types. Note that white-box approaches are just tools. To actually improve the SLS algorithm performance, the algorithm designer must gain and then utilize the obtained insights.

3.6 Summary

1. Although it is easy to come up with a working SLS algorithm to solve a particular COP, designing and tuning an SLS algorithm to achieve consistently good results within a short amount of development and running time is not so straightforward. We call this problem the SLS Design and Tuning Problem (DTP).

2. We classify the SLS DTP into three sub-types:
   Type-1: Calibrating parameter values
   Type-2: Choosing best components, and
   Type-3: Adding search strategies

3. SLS DTP is unavoidable as we cannot change the COP being solved so that it suits our SLS algorithm. We must address this SLS DTP in order to create a good SLS algorithm for the given COP. Experts in the field (as quoted in Section 3.4) agree on the importance of having this issue addressed.

4. We classified the approaches to address the SLS DTP into black-box approaches (treat the SLS runs as black-box) or white-box approaches (analyze the details of SLS runs to obtain insights).

   (a) Black-box approaches: Meta SLS, CALIBRA, F-Race, ParamILS, and GGA.

   (b) White-box approaches: Descriptive Statistics, Inferential Statistics, Human-Guided Search, and Visualization of Search Landscape and/or Behavior.

   Each approach has its own strengths and weaknesses.

3.7 Looking Ahead

We have reviewed various approaches to address this SLS DTP, but no approach is clearly superior to the others in addressing all types of SLS DTP.

In Chapter 4, we propose a better and generic white-box visualization technique called Fitness Landscape Search Trajectory (FLST) visualization. With this FLST visualization, one can better understand his SLS algorithm and use his intelligence to get insights to address the SLS DTP. In Chapter 5, we show an SLS visualization tool Viz that implements this FLST visualization.

Chapter 4

Fitness Landscape Search Trajectory Visualization

"It is not who I am underneath but what I do that defines me"[1]
— **Bruce Wayne in Batman Begins (2005)**

In this chapter, we discuss Fitness Landscape Search Trajectory (FLST) visualization, the main visualization idea in this thesis. FLST visualization is used for explaining SLS algorithm behavior on its COP fitness landscape. This chapter combines and updates the materials published in [83, 56, 59, 60, 57, 61, 58].

4.1 Motivation and Outline

The analysis of SLS algorithm behavior is complex. When an SLS algorithm does not perform well, it may not be easy to identify what to fix. And when it does perform well but a better performance is expected, it may not be easy to identify what to improve.

The emergence of white-box approaches (see Section 3.5.2) has helped the algorithm designers to analyze *some* issues in their SLS algorithm behavior. These white-box approaches may also reveal exploitable generic properties of the COP at hand. With the obtained insights, users are in a much better position to take intelligent decisions on how to modify their SLS algorithm and its configuration ($M + \Phi$) in order to obtain a better performance.

In Section 2.3.3, we have shown that an SLS trajectory can be seen as a walk in the COP fitness landscape. It will be intuitive if one can visualize the fitness landscape and understand what the SLS algorithm is doing there, i.e. by answering some of the questions listed in Section 4.2 and 4.3. However, existing white-box approaches have difficulties in answering many of these questions (see Section 4.4).

[1] It is not the SLS algorithm per se, but how its search trajectory behaves on a COP fitness landscape that defines the SLS algorithm.

What is missing in the literature is a powerful *generic* visualization[2] that can answer these questions yet not tied to a particular COP or SLS algorithm. In Section 4.5 and 4.6, we explain our generic Fitness Landscape Search Trajectory (FLST) visualization. This FLST visualization can help users in answering fitness landscape and search trajectory questions and to deal with the main problem in this thesis: the SLS Design and Tuning Problem (DTP).

4.2 Explaining the Fitness Landscape of a COP Instance

We begin with an analogy. Let's imagine real-life landscapes: mountain ranges, city skylines, or sandy deserts. COP fitness landscapes are analogous: points (solutions) scattered in a kind of landscape and each point has different height (solution quality). It is just that these COP fitness landscapes are not in 3 dimensions but in n dimensions, where n is the COP instance size.

To understand the COP fitness landscape traversed by the SLS algorithm, researchers typically resort to analyze its features only. This is more manageable. For example: local optima distribution, local optima variability, basin of attractions, position types, etc (see Chapter 5 of [68]). Typically, SLS algorithm designers pick COP instances and ask some questions about their fitness landscapes such as the ones below:

1. **What does the local optima distribution look like?**

   (a) If local optima are clustered in one big cluster (the 'Big Valley' property), then the typical strategy is to concentrate the search effort. [45, 89, 129, 59, 61]

   (b) If local optima are spread out (no major cluster), then the typical strategy is to perform more diversification. [129, 89, 95, 57, 61]

2. **What is the solution quality distribution (variability) of local optima?** Four possible scenarios:

   (a) The variability of local optima is very low (plateau fitness landscape) and has many regions have solutions with similar Objective Value (OV).
   Further questions to be asked are:

   - Can the objective function be enhanced to differentiate solutions better?
   - Is it better to use an exact algorithm instead? [82]

   (b) The variability of local optima is low (smooth fitness landscape) and the quality differences between global and any local optima are small.
   A further question to be asked is:

   - Does more intensification help? [57, 61]

[2] In this thesis, we mainly use visualizations because it is generally easier to understand large volumes of information if they are presented graphically in an effective fashion. These visualizations exploit *human visual strengths* in understanding *spatial information*, in associating *colors, shapes* with information, in observing *trends*, and in detecting *patterns, anomalies, outliers, discontinuities*.

(c) The variability of local optima is high (rugged fitness landscape) and the quality differences between global and any local optima are large.
A further question to be asked is:

- Does more diversification help? [129, 95, 126, 57, 61]

(d) The global optima are isolated (fitness landscape with 'Golf-Holes') and most local optima have poor quality, except the global optima.
Further questions to be asked are:

- Can some fitness landscape properties or problem-specific knowledge be used to help guiding the search in this difficult fitness landscape? [93, 58]
- Is it better to use an exact algorithm instead? [87]

The answers for these questions can be good starting points for designing appropriate SLS algorithms. This is because walking through different fitness landscape types may require different search strategies.

4.3 Explaining SLS Trajectories on a COP instance

After exploring how the fitness landscape looks like, the algorithm designer now needs to understand how his SLS algorithm behaves on this fitness landscape. Typically, he will want to ask some combination or all of the following questions:

1. **Does the SLS algorithm behave as what was intended?**

   (a) Is it attracted to good regions?
   (b) When it is designed to search around the 'Big Valley' region, does it wander too far from any local optima that it finds along the search trajectory?
   (c) When the fitness landscape is very rugged, is the stronger diversification used effective to help it escape from deep local optima?
   (d) When its search strategy is to focus on exchanging short edges in TSP tour (a problem-specific heuristic), does it exchange too many long edges?

   Note: Sometimes, the SLS algorithm may exhibit behavior well beyond the original intent of its design, coined as 'failure modes' by [153].

2. **How good is the SLS algorithm in intensification?**

   (a) Does it have sufficient exploration within a local neighborhood?
   (b) Does it stay around a good region 'long enough' before it attempts to make escape moves from that region?

   Note: If the intensification is not sufficient, the SLS algorithm may find a better solution sometime later in the future after revisiting the same region R that has been visited in the past just because it did not intensify enough around that region R previously. Thus, it will take longer to find the solution.

3. **How good is the SLS algorithm in diversification?**

   (a) Does it make successful *non-local* moves to previously unexplored parts of the fitness landscape?
   (b) Does the diversification manage to lead the search towards a new (hopefully better) Best Found BF solution or even the Global Optima GO?

   Note: Most of the time diversification will fail to bring the SLS algorithm to good region as it usually destroys good elements of the current solution. However, the goodness of diversification should be measured by its potential in helping the SLS algorithm to break out from stagnation, e.g. if it manages to help the search 1-2 times (and fails 100 times) throughout the search, it can still be considered useful.

4. **Is there any sign of cycling behavior?**

   Note: One main problem faced by SLS algorithm is in escaping local optima.

5. **Where in the fitness landscape does the SLS algorithm spend *most* of its running time?**

   (a) The SLS algorithm mostly searches in good regions.
   (b) The SLS algorithm mostly searches in bad regions.
   (c) The SLS algorithm mostly searches in regions far from the region that contains the BF solution?
   A further question to be asked is:
      - Is the BF solution found due to a random move or because the SLS algorithm is progressively narrowing its search to it?

   (d) The SLS algorithm mostly searches in the region near the initial solution.
   A poor SLS algorithm may be unable to escape from the first local optimum.
   (e) The SLS algorithm is trapped in a deep local optimum region.
   The search is progressing well until it arrives at a particular local optimum region at a certain iteration. After that, it stays there for a large number of iterations.

   Note: Searching in the 'wrong place' is another common issue of SLS algorithms.

6. **How does the SLS algorithm manage to find the Best Found BF solution?**

   (a) The search quickly focuses on a promising region and it is just a matter of time before it eventually arrives at BF solution.
   (b) The search arrives at BF solution only after searching for a long time and most of the time it is far from the location of BF solution.
   A further question to be asked is:
      - Is this a lucky diversification?

(c) The search arrives at BF solution early in the search and then stagnates.
A further question to be asked is:

- Is the quality of this BF solution close to the Best Known BK solution? If not, this search is very poor.

7. **How the important solutions (e.g. initial solution s^0, BF) fare w.r.t global optimum/best known solution in terms of quality and distance?**

   For the initial solution s^0, this information determines the quality of the construction heuristic. For the BF solution, this information determines the quality of the overall SLS algorithm.

8. **How wide is the SLS algorithm's coverage?**

   (a) Even after long runs, the diversity of solutions in ST is low.
   A further question to be asked is:

   - Do we need a stronger diversification?

   (b) The diversity of of solutions in ST is ok, but global optima/best known solutions are missed.
   A further question to be asked is:

   - Do we need to search on different areas than currently explored?

9. **What is the effect of modifying a certain configurable part (parameter, component, or strategy) w.r.t the SLS algorithm behavior?**

   (a) The performance improves, e.g. now the new SLS run manages to find a better BF solution.

   (b) The performance deteriorates, e.g. now the new SLS run does not manage to find previously found BF solution.

   (c) There is no significant performance difference as the effects are not obvious! Further questions to be asked are:

   - Is the SLS algorithm quite robust to handle such changes?
   - Is it because the modified part is not significant to the search?

4.4 Limitations of Current White-Box Approaches

Answers to the questions posed in Section 4.2 and 4.3 are hard to obtain because:

- It is known that the fitness landscape size is very large. Exploring the entire search space is not feasible.
- We are often unable to obtain the answers a priori without running the SLS algorithm and then analyzing its empirical results using white-box approaches.
- Different instances of the same COP may have different fitness landscapes [68], perhaps due to the COP characteristics or instance sizes.

- The SLS behavior depends on the fitness landscape [68, 89, 126]. SLS algorithm with a certain configuration may behave differently on different fitness landscapes.
- The selected configuration (parameter values [17, 3], components [24], and search strategies [56]), implementation details, and any bugs, determine the *actual* SLS algorithm behavior. This behavior may be undesirable, e.g. doing diversification when intensification is expected (the 'failure modes' [153]).
- Stochastic elements in the SLS algorithm imply that SLS runs can be different when replicated.

In Section 3.5.2, we have discussed existing white-box approaches for analyzing the COP and/or the SLS algorithm. Unfortunately, although each white-box approach can be used to reveal *some* information, they still have difficulties. For example:

- FDC analysis can detect some COP fitness landscape characteristics, e.g. "whether a 'Big Valley' exists?", but it is not designed to describe its details, as shown later in Figure 4.8. FDC analysis is also not designed to answer SLS algorithm behavior questions, e.g. "how does the SLS algorithm manage to find the BF solution?".
- On the other hand, the RTD analysis can detect some SLS algorithm behavior like search stagnation but it is inadequate to answer questions like "where in the fitness landscape does the SLS algorithm spend most of its time?", as shown later in Figure 4.13.
- In Figure 3.1.A, we can observe the solution quality distribution but cannot answer other questions, e.g. "How wide is the SLS algorithm coverage?".
- In Figure 3.3, left, we have a visualization of a 2-dimensional COP that only works on 'toy' COPs with 2 variables. The visualization cannot be used for typical COPs which have $n \gg 2$.
- In Figure 3.3, right, we have a multidimensional scaling approach that maps a small number of solutions of 10-dimensional data set into 2-dimensional. This approach is not scalable for typical COPs with $n \gg 2$ and thousands of iterations.
- In Figure 3.4, we have another mapping of the n-dimensional space into 2-D that can partially answer questions like "How wide is the SLS algorithm coverage?" but not really successful due to the cluttered visualization when n is large.
- In Figure 3.5, we have constraint-specific visualization. But it is hard to analyze the SLS algorithm behavior just by looking at the constraints' values.
- In Figure 3.6, we have another attempt of visualizing fitness landscape. However, this visualization does not show SLS trajectory information.

Even the combination of *all* existing white-box techniques is not effective to really answer some questions posed in Section 4.2 and 4.3 above, e.g. "Does it behave as what we intended?', "How good is the SLS algorithm in intensification or diversification?", or "Where in the fitness landscape does the SLS algorithm spend most of its time?".

In this chapter, we propose to overcome this inadequacy by using our Fitness Landscape Search Trajectory (FLST) visualization, which is capable to display *approximate* COP fitness landscape structure and animating the SLS trajectory on it. As in other white-box approaches, we leverage on the human's strengths to gain insights from the visualization (also see Appendix C).

4.5 Fitness Landscape Search Trajectory Visualization

Given sufficient time and memory, we can explore the entire solutions in the COP fitness landscape FL, which corresponds to the situation where we have perfect information. Then, we can explain *any* COP FL structure and *any* SLS algorithm behavior on the FL precisely as we have all the information. However, exploring all solutions in the FL is not feasible. Thus, we introduce a more feasible way to analyze SLS algorithm behavior *without* enumerating the entire FL. We give an *overview* of our approach via an illustration below. The technical details of this FLST visualization is given in Section 4.5.2 onwards.

4.5.1 Illustrating FLST

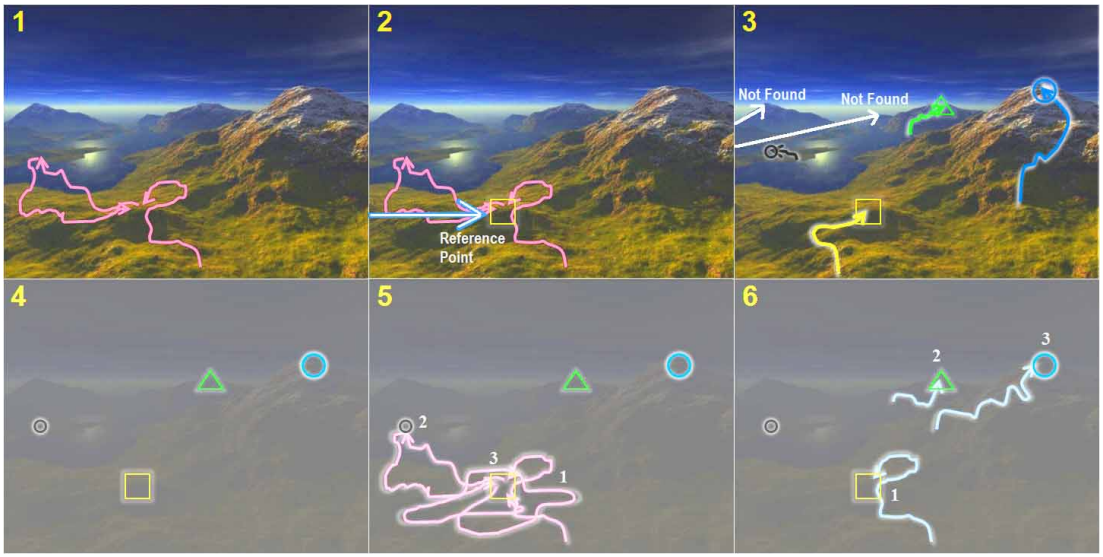


Figure 4.1: Analogy of Finding Highest Mountain

In Section 2.3.3, we show that SLS algorithm can be understood as a walk on a COP's fitness landscape. Figure 4.1 part 1 is a direct visualization of this analogy. The mountain range in the background is an analogy of fitness landscape $FL = \langle S(\pi), d(s_1, s_2), g(s) \rangle$. The search space of COP instance π, the $S(\pi)$, is visualized as a collection of points in the figure. An appropriate[3] distance function $d(s_1, s_2)$ (e.g. Hamming/bond distance) separates those points[4]. The objective function $g(s)$ deter-

[3]In Chapter 7, we explain which distance function is selected for each of the COP in our experiments.

[4]For FLST visualization purposes, the definition of FL in Section 2.3.2 uses $d(s_1, s_2)$ instead of $N(s)$.

mines the height of each point on the mountain range. Global Optima GO are the highest mountain(s) and Local Optima LO are the other mountains.

Now look at the pink search trajectory ST in the same Figure 4.1 part 1. We observe a 'movement' on this mountain range, but may not be able to describe it. Now consider Figure 4.1 part 2. If we regard the yellow rectangle as a reference point, we can now say more about the pink search trajectory, for example: "The search trajectory first hits the yellow rectangle solution, then it moves somewhere else, then after a certain number of iterations, it hits the yellow rectangle solution again. Is this a solution cycling phenomenon? Is the SLS algorithm trapped?".

Suppose now we run one *or more* heuristic(s)/SLS algorithm(s) possibly with different configurations, and perhaps by including some form of random walk. These different runs, due to their heuristic and stochastic nature, may visit different regions in the fitness landscape. We collect a number of samples of reference points (mountain peaks) that are scattered in the fitness landscape, as illustrated in Figure 4.1 part 3. We name this reference point the **'Anchor Point'** (AP).

Of course, we should expect to miss *some* good points due to the incompleteness of the SLS algorithms used to collect them (observe the two white arrows in Figure 4.1 part 3 that point at two mountain peaks in the background that are 'Not Found'). But if we collect appropriate APs (elaborated in Section 4.5.2), we can have an *approximation* of the fitness landscape. In Figure 4.1 part 4, we see a dimmed background with four APs only. Each AP is color+shape labeled to indicate their quality: **Good/O**, **Medium/△**, **Bad/□**, **VeryBad/.** (see Figure 4.5 for details). In this example, we can approximate the actual fitness landscape with just four APs.

Now we can playback the SLS algorithm and give a more meaningful description about its search trajectory. At time t, we measure the distance of solution $s^t \in ST$ w.r.t known APs using an appropriate distance function. We repeat this process from $t = 0$ (initial solution) until $t = lastItr$.

For example in Figure 4.1 part 5, we can say that the pink trajectory visits a **Bad/□** AP (see label 1), goes to a **VeryBad/.** AP (label 2), and then cycles around these APs (label 3). It fails to reach the **Medium/△** or **Good/O** APs. We can say that such a search behavior is bad.

In Figure 4.1 part 6, we can say that the **blue** trajectory performs some diversification after hitting an AP (which is a local optima). It manages to reach the **Bad/□** AP (label 1) plus the **Medium/△** (label 2) and **Good/O** APs (label 3). We can say that it performs better than the pink trajectory shown in Figure 4.1 part 5.

By having a good APs and proper presentation techniques, we can *approximately* explain the SLS algorithm behavior as the movement of the search positions from one AP to another AP over time. These APs are used to give a 'semantic description' to SLS trajectories, e.g. 'moving closer' to a good AP_1, or 'getting away' from poor AP_2, or 'never explore' the good cluster containing AP_3, or 'always within distance $< \delta$ units from AP_4 (trapped around a local optimum), etc. This is the essence of the Fitness Landscape Search Trajectory (FLST) visualization.

4.5.2 Anchor Points Selection

Terminologies

Anchor Point AP**:** An AP is a distinguished solution in $S(\pi)$. It can be seen as a reference or signpost in the fitness landscape. FLST visualization explains the SLS trajectory movement using these APs.

Anchor Point Set $APset$**:** $APset \subset S(\pi)$. $|APset|$ denotes the size of the $APset$. APs in $APset$ are sorted based on solution quality, i.e. AP_0 is the Best Found BF solution.

Motivation

The first important step in building a Fitness Landscape Search Trajectory (FLST) visualization is to select a *relatively small* number of APs out of a very large number of points in the original n-dimensional fitness landscape. This is because it is not feasible to visualize *all* the points in the fitness landscape. The selected APs form a set called $APset$. There are several natural questions that arise because of this requirement.

- Which kind of points should we select?
- How to obtain these points?
- What is the limit of number of points that can be selected?

These design questions must be addressed as $APset$ influences the resulting FLST visualization. In this section, we discuss the trade offs of each our design choices.

Local/Global Optima for $APset$

The objective function is the major driving force used by SLS algorithms to navigate the COP fitness landscape. As (a good) SLS algorithm is attracted to and spends more time around points with better Objective Value (OV), the most appropriate points to be kept as APs (the reference points) for the FLST visualization are the ones with good solution quality.

We choose *local optima* (global optima are also local optima) to be included in the $APset$ for two reasons. First, each local optimum has OV superior to its neighbors. Second, when the current search trajectory is *close* to a local optimum X, it is likely that the search trajectory is now inside the *region* containing X (the 'basin of attraction' of X). If X is in the $APset$, we can describe the SLS trajectory near (around) local optimum X.

This design choice means that the fitness landscape in FLST visualization only shows (selected) local optima information only. Points in SLS runs that are not local optima will be shown using another approximate visualization as discussed in Section 4.5.4.

Collecting Local Optima from SLS RunLogs

We decide to utilize the SLS algorithm itself to collect local optima. When the SLS runs, we set it to record the solutions that it finds. We may run the same SLS algorithm with different configurations or use other SLS algorithms – including random walk. Due to different heuristics and stochastic behavior, these SLS runs are likely to traverse different parts of the fitness landscape.

These SLS runs produce streams of n-dimensional solutions. Although we do not determine for sure whether a solution is a local optimum according to its definition (w.r.t its neighborhood), we can know which ones are not by utilizing the local move property. If solutions s^{t-1}, s^t, and s^{t+1} are *adjacent* solutions in the search trajectory of an SLS algorithm for *minimizing* COP and $g(s^{t-1}) \geq g(s^t) \leq g(s^{t+1})$, then we know that s^t is a *potential* local optimum (called $PotAP$) while s^{t-1} and s^{t+1} are not.

We choose to select the points using SLS algorithms to minimize the additional efforts that must be taken by the user of this FLST visualization. We do not require the user to create additional algorithms to obtain local optima more systematically – this defeats the original purpose of creating FLST visualization for addressing the SLS Design and Tuning Problem. The user of our FLST visualization only needs to tweak the SLS algorithm to be analyzed to record the stream of solutions. However, this design choice limits the FLST visualization to show explored landscape only.

Limiting the Size of the $APset$

After filtering the obvious non local optima from SLS runs, we still face a large number of potential local optima ($PotAP$), especially for COP with larger instance size n. One may argue that we can use 'zoom and pan' features to visualize larger instances. However, this is not scalable as the screen resolution for displaying the FLST visualization is relatively small, e.g. 1280x1024. Moreover, observing too many tiny APs when the FLST visualization is zoomed out may not be intuitive. Therefore, we decide to have a *small* and *constant* $|APset|$. The appropriate $|APset|$ depends on the COP being solved and the corresponding SLS algorithm used. This will be discussed later in Chapter 7.

This design choice means that the FLST visualization is unable to show many local optima that are not selected in $APset$. Thus, we cannot show the Search Trajectory information during the times ST is near the unselected potential local optima.

Increasing the Diversity of the $APset$

Given that we can only select a limited number of APs to explain the fitness landscape and search trajectory on it, we need to select APs that can tell us more about the fitness landscape and the search trajectory. We also need to have a diverse $APset$ on top of high quality $APset$ by selecting local optima, as elaborated below.

Later in Section 4.5.4, we use a distance function to measure how close the current solution s^t at time t in the search trajectory w.r.t the APs in $APset$. When two

solutions: s^t and anchor point X are close to each other, i.e. $d(s^t, X) \leq \delta$ for a small δ, then we can say that s^t may be inside the basin of attraction of AP X. But if $d(s^t, X) > \delta$, then the AP X is not useful for explaining what is happening with s^t.

To make $d(s^t, X) \leq \delta$, $APX \in APset$ for many events in ST (especially for important events like "stuck in a good local optimum" or "approaching the BF solution"), we need to pick APs that are diverse enough so that the APs cover a *wide* part of the fitness landscape. We want to maximize the number of points s^t in the SLS run where there exists an AP X which is close to s^t by a certain (small) δ.

Formal Definition of the AP Selection Problem

In summary, we need to select diverse and high quality APs from potential local optima. To capture these requirements, we define the AP Selection Problem as a multi-objective optimization problem: obtain an $APset$ such that the $AverageQualityGap$ is minimized and the $DiversityIndex$ is maximized. Note that $AP_i, AP_j \in APset$.

$$AverageQualityGap = \frac{\sum_{i=0}^{|APset|-1} |(g(AP_i) - g(BK)|}{|APset| * g(BK)} \tag{4.1}$$

$$DiversityIndex = \frac{\sum_{i=0}^{|APset|-2} \sum_{j=i+1}^{|APset|-1} d(AP_i, AP_j)}{|APset| * (|APset| - 1)/2 * n} \tag{4.2}$$

$AverageQualityGap$ (AQG) measures the average gap (in percentage-off) between the quality of APs in $APset$ w.r.t the Best Known (BK) OV. $DiversityIndex$ (DI) measures the average distance between any two APs in $APset$ and ranges from 0 (no diversity – only 1 AP) to 1 (maximum diversity – all APs are very different). These two criteria are conflicting.

Heuristics for the AP Selection Problem

There are $O(C^{|ST|}_{|APset|})$ ways to choose $|APset|$ AP from $|ST|$ points in an SLS run ST. An exact algorithm may take too long. In practice, we do not need an optimal $APset$ for FLST visualization as long as the visualization is meaningful to the human user. A fast method that can process thousands of potential local optima ($PotAP$) efficiently is preferable. Therefore, we adopt heuristic approaches to get a fast and reasonable result for the AP Selection Problem.

One simple heuristic is to *greedily* select the top-$|APset|$ $PotAP$. This heuristic runs in $O(|ST| * \log(|APset|))$ per SLS run by maintaining the top-$|APset|$ $PotAP$ with a heap data structure. This heuristic produces low $AverageQualityGap$ but also produces a low $DiversityIndex$ as the selected APs will be mostly the local optima around the region containing the BF solution – this is not good[5].

Another simple heuristic is to *randomly* select $|APset|$ $PotAP$. This heuristic can be implemented in $O(|ST|)$ per SLS run by randomly decides whether a certain $PotAP$

[5]This works for COP with the 'Big Valley' property (see Section 7.2) since most parts of the SLS runs are within this region. But for COPs with diverse multi modal local optima, we will lose the representative for a lot of other local optima regions which are far from this BF region. If this happens, we cannot analyze the search trajectory when it is not in the BF region yet.

should be kept (thus replacing another existing AP in $APset$) or ignored throughout the SLS run. This heuristic is expected to produce an $APset$ with reasonable $DiversityIndex$ but usually bad $AverageQualityGap$.

These two heuristics are inadequate for addressing the AP Selection Problem. Thus we develop our `AP-Selection` heuristic described below.

AP-SELECTION($Seed, SLSRun, TargetAPsize$)

```
1  InitRandom(Seed) // For breaking ties randomly (WorstAP/WorstNearestAP)
2  APset ←load previous APset (if any)
3  for each PotAP ∈ SLSRun // iterate through all PotAP in SLS run
4      do if Size(APset) < TargetAPsize // the first few PotAP are taken
5            then Add PotAP to APset if PotAP is not already ∈ APset
6            else  if BetterSolutionQuality(PotAP, WorstNearestAP)
7                     then PotAP replaces WorstNearestAP
8                     else  if EqualSolutionQuality(PotAP, WorstNearestAP)
9                              then PotAP replaces WorstAP
10 Sort APset based on solution quality, with AP_0 as the Best Found AP
11 Save APset
12 return APset
```

Line 3-7 of our `AP-Selection` heuristic is essentially similar to the heuristic that greedily selects top-$|APset|$ $PotAP$. But in line 6-7, instead of comparing the solution quality of current $PotAP$ with $WorstAP$ that potentially decrease $DiversityIndex$, our heuristic compares the solution quality of current $PotAP$ with the $WorstNearestAP$ (from that $PotAP$). $WorstNearestAP$ is found by first identifying the AP currently in $APset$ that has nearest distance with current $PotAP$ (random tie breaking is used when necessary). The rationale is that an SLS run is a stream of solutions. Thus, the distance between immediate solutions (s^t and s^{t+c} for a small c) in an SLS run is usually small. Rather than keeping both in $APset$ if both are good local optima, this heuristic *tends* to keep only one the better $PotAP$ for each region in fitness landscape as it dominates other APs with worse solution quality *near* the chosen one.

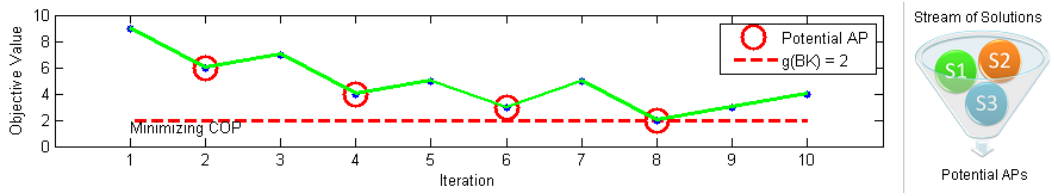
Special case: Line 8-9 are used if the current $PotAP$ has similar solution quality to the $WorstNearestAP$. This may happen if the fitness landscape is plateau or when there are multiple local (or global) optima that have similar solution quality. In either case, we prefer to visualize this important information even if it causes a decrease in the $DiversityIndex$. If this happens, then $PotAP$ will replace the $WorstAP$ currently in the $APset$ (random tie breaking is used when necessary).

Line 2 (load) and Line 11 (save) are used to improve the solution quality of $APset$ across various SLS runs. Rather than starting from scratch – which will make FLST visualization looks different every time – $APset$ of a COP instance is *updated* when a new SLS run is performed on that instance. Line 6-9 dictates that only new $PotAP$s from a new SLS run with better (or equal) solution quality can replace older APs currently in the $APset$. This allows us to have more stable FLST visualization and

let us explain the behavior of an SLS run with some APs found from another (either better or worse) SLS run(s).

The time complexity of our `AP-Selection` heuristic is $O(|ST|*|APset|*n)$, heavier than the greedy or random selection heuristics. The big part is the $O(|APset|*n)$ effort to find the important $WorstNearestAP$. We decide to use the `AP-Selection` heuristic because it is biased towards getting better local optima (decrease $AverageQualityGap$) but has a better $DiversityIndex$ than the greedy selection heuristic.

To illustrate the differences of the three heuristics mentioned in this section, we use the following simple example (see Figure 4.2). There are 10 consecutive bit strings $\in ST$ with length $n = 6$ and different OVs. We want to select 3 bit strings only. Random selection heuristic randomly selects – for example – the points at iteration 3, 6, 7. Greedy selection heuristic selects the top-3 points with good solution quality: the points at iteration 6, 8, 9. Our `AP-Selection` heuristic initially selects points at iteration 2, 4, 6 as they are the first 3 $PotAP$. Later, it replaces the AP from iteration 6 with the AP from iteration 8 because AP from iteration 6 (`011010`) is the $WorstNearestAP$ from $PotAP$ from iteration 8 (`011001`). The $AverageQualityGap(AQG)$ and $DiversityIndex(DI)$ of the three heuristics for this example is shown in Table 4.1.



Iter	Point	OV	Gap	Iter	Point	OV	Gap
1	100100	9	7	6	011010	3*	1
2	100110	6*	4	7	011011	5	3
3	000110	7	5	8	011001	2*	0
4	000010	4*	2	9	111001	3	1
5	010010	5	3	10	111011	4	2

Figure 4.2: Collecting Potential APs (**Red Circles O**) from an SLS run that flips one bit per iteration. For column 'OV', lower value is better (minimizing) and star (*) indicates 'Potential AP'. Column 'Gap' is column 'OV' minus $g(BK)$.

	Random Selection	**Greedy Selection**	`AP-Selection` **Heuristic**
$APset$	3, 6, 7 (Random)	6, 8, 9 (Top-3)	2, 4, 8 (8 replaces 6)
AQG	$\frac{5+1+3}{3*2} = 1.50$	$\frac{1+0+1}{3*2} = 0.33$	$\frac{4+2+0}{3*2} = 1.00$
DI	$\frac{3+4+1}{3*6} = 0.44$	$\frac{2+3+1}{3*6} = 0.33$	$\frac{2+6+4}{3*6} = 0.67$

Table 4.1: Comparing AP Selection Heuristics.
$AQG = AverageQualityGap$ and $DI = DiversityIndex$.

Table 4.1 illustrates that:
Random selection produces $APset$ with average DI but bad AQG;
Greedy selection produces the lowest AQG but with low DI too – this will make many points in search trajectory too far from $APs \in APset$;
Our `AP-Selection` heuristic is biased towards low AQG but has better DI than greedy.

4.5.3 Fitness Landscape Visualization

Motivation

After we have collected reasonable high quality and diverse *AP*s, the next important step is to present these *AP*s in a user-friendly visualization.

The appropriate dimension to visualize movement on a landscape is 2-D since humans are good in discerning 2-D spatial information[6]. When there are only 2 decision variables, it is easy to plot a 2-D graph (see Figure 3.3, left). However, since the size of a combinatorial solution is usually $O(n)$, e.g. a permutation of n items, a bit string of n items, etc, visualizing this point in high-dimensional space in 2-D is a challenge.

Formal Definition of *AP* Layout Problem

We introduce a novel *2-D visualization space* VSPACE to represent a fitness landscape. This is not a projection from n-dimensional space into 2-D (compare with Figure 3.4, right) which may cause many collisions of points in the 2-D space. In VSPACE, the x and y-axis are not related to the variables, rather they are meant to make it easier to see the search trajectories in a 2-D layout. We make use of a generic distance function to measure the pairwise distance between these *AP*s and a layout algorithm to layout these *AP*s in VSPACE.

We remark that FLST visualization requires that we use distance function that satisfies the triangle inequality property as it corresponds to spatial intuition. Examples of such distance functions are the Hamming and bond distance [123, 124]. Using a distance function that does not satisfy the triangle inequality property may distort the FLST visualization.

This *AP* Layout Problem can be seen as a kind of graph drawing problem the where 2-D positions of the *AP*s should *approximately* reflect the distance between these *AP*s in n-dimensional space. However, this layout is usually imprecise, as elaborated below.

A perfect 2-D layout for *AP* layout problem cannot be achieved in general. The reason is that the distance function used to measure the difference between two combinatorial solutions is usually *not* the 2-D Euclidian distance. As an illustration, consider the following *AP*s that are represented as bit strings:

$AP_a = 0000$ $AP_b = 0111$ $AP_c = 1011$

Their Hamming distances are:

$d(AP_a, AP_b) = 3$ $d(AP_a, AP_c) = 3$ $d(AP_b, AP_c) = 2$

One 2-D layout of AP_a, AP_b and AP_c is a triangle of length 3, 3, 2 as shown in Figure 4.3.A. This is a perfect 2-D layout since it preserves the *AP* distances.

Now suppose that there is one more anchor point, $AP_d = 1111$. The Hamming distances from AP_d to the rest are:

$d(AP_d, AP_a) = 4$ $d(AP_d, AP_b) = 1$ $d(AP_d, AP_c) = 1$

[6] We live in a 3-D world but computer screen is essentially 2-D. Drawing 3-D data in a 2-D screen (e.g. [86]) causes foreground objects to obscure background objects. 2-D data are naturally easier to comprehend. Dimensions higher than 3-D are not natural, more complex, and difficult for visualization.

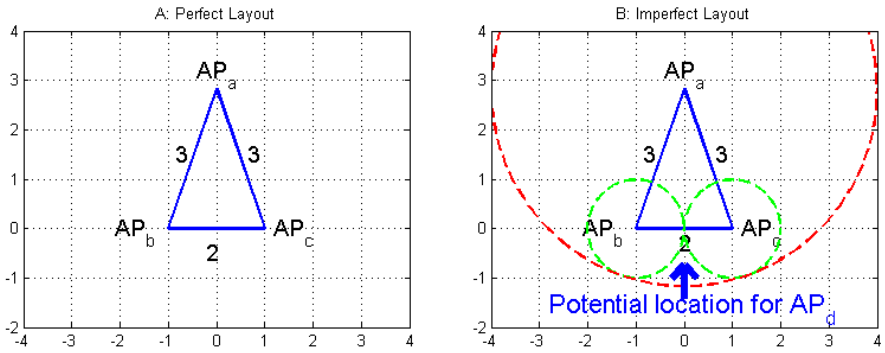


Figure 4.3: Perfect Layout is Hard to Attain

Now, it is not possible to draw all the four APs in 2-D such that the 2-D Euclidean layout distance preserves the Hamming distance. In Figure 4.3.B, we draw circles on each AP_a, AP_b, AP_c to indicate their distance w.r.t AP_d. It is apparent that there is no common intersection point between all the three circles. If an imperfect layout is acceptable, the point highlighted by the blue arrow indicates a potential location of AP_d with small layout error.

If the $DiversityIndex$ is higher, we will have a harder layout problem. For example, replace AP_d with $AP_e = 1100$ with $d(AP_e, AP_a) = 2$, $d(AP_e, AP_b) = 3$ and $d(AP_e, AP_c) = 3$. The layout error is larger than the one shown in Figure 4.3.B.

As it is not possible to attain a perfect layout in general, we view this layout problem as another optimization problem. We want an AP layout on VSPACE that minimizes the layout error `errAP`, defined below. `ELD`(AP_i, AP_j) denotes the 2-D Euclidian layout distance between AP_i and AP_j in VSPACE. $d(AP_i, AP_j)$ denotes the n-dimensional distance between AP_i and AP_j in n-dimensional space. $AP_i, AP_j \in APset$.

$$errAP = \frac{\sum_{i=0}^{|APset|-2} \sum_{j=i+1}^{|APset|-1} |(ELD(AP_i, AP_j) - d(AP_i, AP_j)|}{|APset| * (|APSet| - 1)/2 * n} \tag{4.3}$$

Given an AP layout with *low* layout error `errAP`, we can expect that if we pick any two APs and observe that these two APs are laid out near (/far) to each other in VSPACE, then they should be *approximately* near (/far) to each other in the actual n-dimensional space, and vice versa. Here, we utilize gestalt principle of proximity [152] where APs that are close to each other are readily perceived as being clustered.

But when AP layout has substantial `errAP`, the user must be careful with the feature mentioned above as there are cases where two APs are laid out near to each other in VSPACE but they are actually quite different in n-dimensional space.

Heuristic for AP Layout Problem

We need a layout algorithm that can reduce the layout error `errAP` efficiently (any good-enough layout is sufficient) and consistent manner (produce the same layout given the same $APset$). There are several proposals for addressing the graph drawing (layout)

problem [31]. In this thesis[7], we use the 'spring model' heuristic layout [77] that falls into the category of force-directed layout[8]. Rather than re-implementing the algorithm, we use the 'spring model' layout algorithm implementation found in the graph drawing tool 'NEATO' [54]. This tool accepts an input graph and produces an output layout according to the 'spring model' layout algorithm. Our wrapper code is shown below.

AP-LAYOUT($APset$)

1 Prepare a graph G with $|AP|$ vertices
2 **for** each pair AP_i and $AP_j \in APset$
3 **do** Set the length of edge (AP_i, AP_j) in G to be $d(AP_i, AP_j)$ // virtual spring
4 Put a constraint that vertex AP_0 in G (BF AP) is to be placed at $(0, 0)$
5 // run external tool 'NEATO'
6 $layout \leftarrow$NEATO$(Seed, maxIter, G)$
7 $Coordinate = parse(layout)$ // read the layout produced by 'NEATO'
8 **return** $\{Coordinate(AP_0), Coordinate(AP_1), \ldots, Coordinate(AP_{|APSet|-1})\}$

Line 1-4 are pre-processing steps. These steps place a spring-like force for every pair of nodes (AP_i, AP_j) where the ideal length of each spring is proportional to the n-dimensional distance between AP_i and AP_j, i.e. $d(AP_i, AP_j)$. In line 4, we put a constraint for 'NEATO' to draw AP_0 (the BF solution which can be either the global optima or the BK solution) in the middle of VSPACE – the *center of attention*. In line 5-6, we run 'NEATO' [54]. 'NEATO' first randomly scatters the APs in VSPACE (it is deterministic given a fixed $Seed$) and then employs a 'spring layout' algorithm that forces each spring to return to its natural length when stretched (drawn as **red dashed line** in Figure 4.4) or shrunk (drawn as **blue solid line** in the same figure). 'NEATO' reduces the overall spring tension[9] modeled in Equation 4.3 (drawn as **green lines** in the same figure). In line 7-8, we parse the output produced by 'NEATO' and return the coordinates of APs in VSPACE.

The springs between all pairs of APs and the fixed position of AP_0 restrict the APs to be laid out by the 'spring layout' algorithm within an imaginary circle around AP_0 with radius $\approx n$ of the COP instance[10]. See Figure 4.9.A for details.

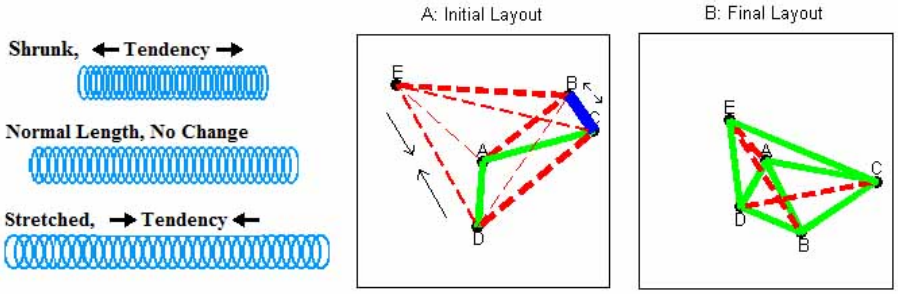
In Figure 4.4, we illustrate an example of running 'NEATO' on 5 points {A, B, C, D, E} where point A is fixed in the center of the screen. In the initial (random) layout, we see that point B is too close to C but too far to {A, D, E}. 'Spring layout' will adjust these points {B, C, D, E} to obtain the final layout by minimizing the tension in all the ${}_5C_2 = 10$ edges. As 'spring layout' is a heuristic, we remark that the layout produced is a local optimum and usually `errAP != 0`! Although FLST visualization cannot be perfectly accurate, we show in Chapter 7 that FLST visualization indeed

[7]There are other techniques for analyzing high dimensional data such as Principal Component Analysis (PCA). PCA may also be used but it is not directly suitable for our visualization needs.

[8]Relevant work by Pohlheim [111] used multidimensional scaling called SAMMON mapping.

[9]Minimizing the difference between Euclidean and ideal distances between nodes in NEATO is known as *multi-dimensional scaling* to statisticians [54].

[10]The maximum distance between a pair of points is usually $O(n)$ and n is the problem size [89]. The typical size of n of COP instances used in this thesis is not too big (e.g. $n < 100$) such that it is feasible to layout these APs on a computer screen.



Edge	Before (distance normalized)			After (distance normalized)		
(AP_i, AP_j)	ELD	d	$\|diff\|$	ELD	d	$\|diff\|$
01. A-B	0.6	0.5	0.1	0.5	0.5	0.0
02. A-C	0.6	0.6	0.0	0.6	0.6	0.0
03. A-D	0.4	0.4	0.0	0.4	0.4	0.0
04. A-E	0.6	0.3	0.3	0.2	0.3	0.1
05. B-C	0.3	0.7	0.4	0.7	0.7	0.0
06. B-D	0.9	0.5	0.4	0.5	0.5	0.0
07. B-E	0.9	0.8	0.1	0.7	0.8	0.1
08. C-D	0.9	0.8	0.1	0.7	0.8	0.1
09. C-E	1.0	0.8	0.2	0.8	0.8	0.0
10. D-E	0.8	0.6	0.2	0.6	0.6	0.0
	errAP: 1.8 / 10 = 0.18			errAP: 0.3 / 10 = 0.03		

Figure 4.4: An Illustration of Spring Model and AP Layout Error `errAP`

works and can reveal important insights even with its inherent impreciseness.

The time complexity of the `AP-Layout` is $O(|APset| + maxIter)$, as we can control the maximum number of iterations of NEATO.

AP Labeling

The AP layout above only gives distance information between APs. To show solution quality information, we label each AP according to its solution quality (the gap of OV w.r.t Best Known BK OV) using redundant[11] features (color+shape). The shapes are small to avoid cluttering the visualization. The labels are shown in Figure 4.5.

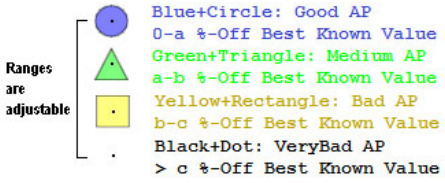


Figure 4.5: The AP Labels

We limit the number of categories to 4 data classes (Good/Medium/Bad/VeryBad) as this is near the limit of human perceptual ability to *quickly* differentiate objects [43]. This AP labeling forms a 'contour map', where the height (solution quality) is easily distinguishable in the map via its color+shape label. The OV ranges for AP classification in Figure 4.5 (the values of a, b, and c) can be interactively adjusted to get more information about the fitness landscape.

[11] To avoid the case where this document is printed in black and white or if the user is color blind.

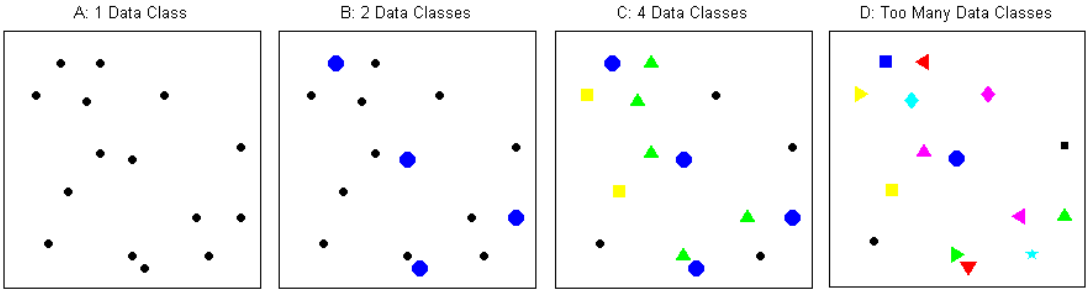


Figure 4.6: *AP* Labeling Enriches the Presentation

In Figure 4.6.A, we observe an *AP* layout on VSPACE *without AP* labeling. Thus, we do *not* know the solution quality information of each *AP*.

In Figure 4.6.B, we set *AP*s that are within 1%-off from *BK* to be labeled with **BlueCircle O**. This way, we can immediately identify that the positions of the good *AP*s are spread out. Here, we are utilizing human perception strength in classifying objects that have similar visual attributes (color, shape, orientation) as belonging to the same group (gestalt principle of similarity) [152].

Of course, we can have a finer grained labeling like in Figure 4.6.C, where we use the 4 data classes described in Figure 4.5 above.

Figure 4.6.D shows that having too many data classes at the same time causes cluttered visualization, which reduces human's ability to understand the visualization. This justifies our limit of 4 data classes.

Fitness Landscape Overview (FLO) Mode

We call this FLST visualization mode the 'Fitness Landscape Overview' (FLO) mode. FLO mode can be used to explain the COP fitness landscape characteristics mentioned in Section 4.2. Local optima distributions are shown by the location of the *AP*s. The solution quality of the *AP*s are represented by the *AP* labels. Circular grid lines[12] are used as a scale to measure the distances between *AP*s especially when `errAP` is low.

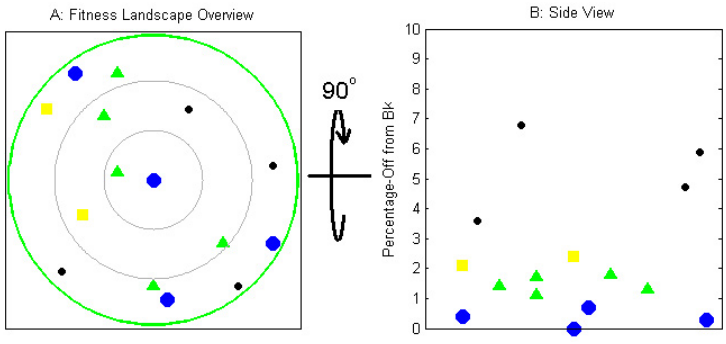


Figure 4.7: Fitness Landscape Overview and Side View Modes

FLO mode is normally viewed from above, i.e. we view the x and y-axis and use the *AP* label to represent solution quality. However, the view can also be rotated 90 degrees

[12]In all our experiments, we set 1 grid line = 10 distance units. **Green border** shows max distance.

along the x-axis to highlight the solution qualities of the $APset$ (now as y-axis). This is called the 'side view' mode. An illustrative example is shown in Figure 4.7.

Comparison with Fitness Distance Correlation (FDC) Analysis

This FLO mode can also be seen as an extension of the FDC scatter plot (see Section 3.5.2), i.e. from comparison of fitness and distance of each local optima w.r.t nearest BF solution only, to comparison of multiple APs. In Figure 4.8, we show an example of what FLO mode can show over FDC scatter plot. In Figure 4.8.A, we see an FDC scatter plot that does *not* exhibit the 'Big Valley' property since there are three good points far from BF solution. With the FDC scatter plot, we *cannot* tell whether the actual distribution actually looks like Figure 4.8.B or Figure 4.8.C. However, this local optima distribution can be naturally shown in the FLO mode, i.e. FDC scatter plot only shows the **red dashed lines** (distances between local optima and BF solution) whereas the FLO mode also visualizes the **green solid lines** (distances between each local optima).

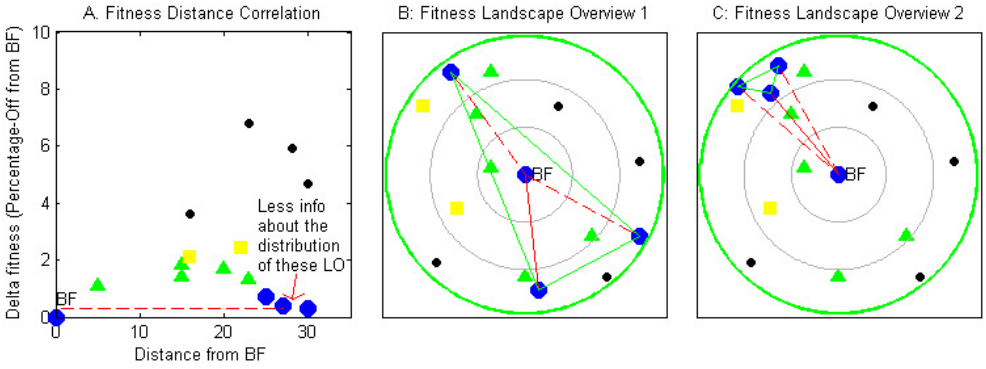


Figure 4.8: Comparison between FDC versus FLO Visualizations

4.5.4 Search Trajectory Visualization

Motivation

FLST visualization does not stop here. To answer the questions about SLS algorithm behavior in Section 4.3, we need to visualize (and animate) the search trajectory on top of the fitness landscape. We can explain the rough search trajectory behavior with just some *minor* additions to the FLO mode shown in Section 4.5.3 above.

Search Coverage Overview (SCO) Mode

The distances between a point s^t, $t \in \{0, 1, \ldots, lastItr\}$ in the search trajectory ST and the APs that have been laid out on VSPACE are used to determine the position of s^t in the VSPACE. If $d(s^t, X) \leq \delta$ for a certain AP X, then a circle of diameter $d(s^t, X) + \epsilon$ is drawn on AP X to show that the current position s^t is within δ units away (or less) from AP X (see Figure 4.9.B). A small ϵ is needed so that when $d(s^t, X) = 0$ (exact match), we see a circle that fits the AP label instead of a dot, which is too small to be

seen clearly. δ is called 'near distance criterion' and adjustable by user, e.g. set $\delta = 0$ to see which APs are *actually visited* or set $\delta = 20\% * n$ to see which APs are either visited or *narrowly passed* by the search trajectory.

Note that there can be more than one $AP\ X \in APset$ that are near to s^t if δ is large enough and there are some APs that are close to each other, i.e. $d(AP_i, AP_j) \leq 2\delta$. For example in Figure 4.9.C, we see that the current point s^t is an exact match with AP_3 (the circle fits) and also quite similar to AP_2 (the circle is larger).

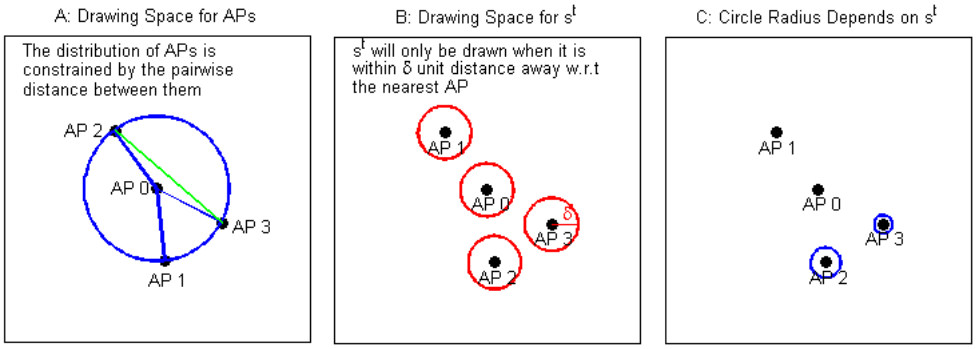


Figure 4.9: The Drawing Space for APs $\in APset$ and $s^t \in ST$

As the SLS algorithm moves locally (unless a strong diversification is performed), adjacent solutions found by the SLS algorithm typically have many similarities: they are from the same region in the fitness landscape. We can set a trail of length l of $s^{t-l}, s^{t-l+1}, \ldots, s^t \in ST$ and draw circles around APs that are within δ distance units with this search trajectory trail. By sequentially advancing the trail step by step over time, we obtain an *animation* of the search trajectory movement on the fitness landscape. We can also see the quality of the region currently being searched via the AP labels of APs with circles drawn on them. If `errAP` is low, the geometric distance of the trail movement can also be used to gauge how 'radical' or 'conservative' the SLS algorithm is at modifying the solutions.

The search trajectory information is not meaningful when s^t is far from all $AP \in APset$. This typically happens if the APs are spread and the SLS algorithm is in 'diversification' phase, or when the SLS algorithm is so poor that it only visits poor solutions outside the current $APset$. If that happens, we do *not* draw any circles to highlight the position of s^t, but we just display the current distance of s^t to the nearest AP (observe Figure 4.10.D where distance indicator – horizontal bar on the top left side – shows that s^t is far from known APs). Thus, when there is no circle appearing on any AP, it means that the search is currently exploring the region far from the recorded $APset$. This is to avoid misleading the human visual perception system. This feature is controlled by the near distance criterion δ mentioned above and we typically set δ to be $\leq 20\% * n$ where n is the instance size/typical maximum distance.

We call this mode the Search Coverage Overview (SCO). See Figure 4.10 for 4 illustrative examples. Suppose the FLO mode has selected, laid out, and labeled the 5 APs $\{A, B, C, D, E\}$. Now, these 4 SLS runs can be described as follows:

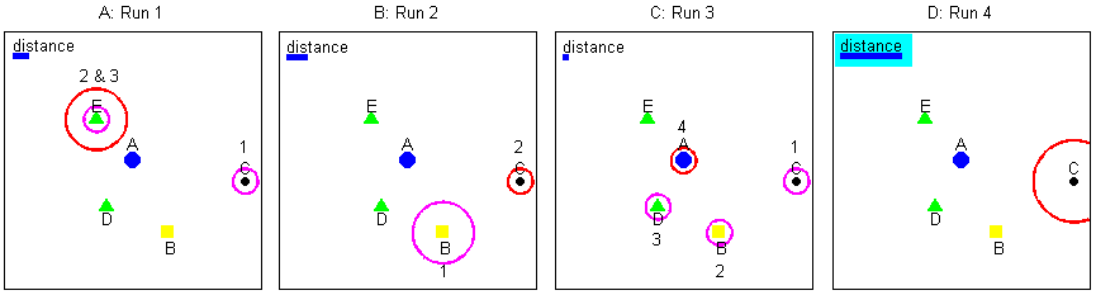


Figure 4.10: Search Coverage Overview Mode, see text for interpretations.

Run 1: CxxxxEFEFE ⇒ SLS algorithm starts from a VeryBad *AP* C (label 1), passes through solutions that are far from known *AP*s (the 'x's), walks to a Medium quality *AP* E (label 2), then **cycles** around *AP* E and its neighbor F (label 3). Neighbor F is not ∈ *APset*, but as it is close to *AP* E, it is drawn as a larger circle around *AP* E. The animation shows a smaller & larger circle appearing alternatingly around *AP* E.

Run 2: GxxxxxCxx ⇒ SLS algorithm starts from a point G which happens to be near Bad *AP* B. This is indicated as a large circle around *AP* B as G is not ∈ *APset* (label 1). But then it walks to a VeryBad *AP* C (label 2). This is a **poor intensification**.

Run 3: CxxBxDxA ⇒ SLS algorithm starts from a VeryBad *AP* C (1), then to a Bad *AP* B (2), Medium *AP* D (3), and Good *AP* A (4). This is a **good intensification**.

Run 4: xxxxxxxxHxxxx ⇒ Throughout the search, the SLS algorithm is mostly not near any known *AP*s. It only briefly pass through a point H near a VeryBad *AP* C and disappear again. It **fails to navigate to promising region** (e.g. D, E, or A).

Note that these are just some possible interpretations of the visualizations shown in Figure 4.10. The actual interpretation will depend on the search strategies being used, problem specific knowledge, fitness landscape structures, etc.

Search Trajectory Detail (STD) Mode

The Search Coverage Overview mode shown above is 'correct' in the sense that if at any time t a circle of radius $\delta + \epsilon$ is drawn on an *AP* X, then the solution s^t is definitely within δ unit distance away from that *AP* X. However, SCO mode is not detailed as s^t can actually be 'anywhere' around the radius of that enclosing circle of *AP* X.

To be more precise, we need to set an approximate position of s^t in the VSPACE. Connecting series of approximate positions of $s^t \in ST$ with lines in an animated fashion is more natural than animating a series of circles in SCO mode. We call this mode as Search Trajectory Detail (STD) mode[13].

As an example, see the SCO mode in Figure 4.11.A. Here, we observe a walk of an Iterated Local Search (ILS) algorithm on a fitness landscape. We can only say that the ILS starts from local optima 1 (LO1), LO2, LO3, and then the ILS reaches the global optima (GO). Compare this with the STD mode in Figure 4.11.B. Here, we observe that from each local optima, the ILS tries to search around it and return to previous

[13]Both SCO (circles) and STD (lines) mode can be active at the same time in FLST visualization.

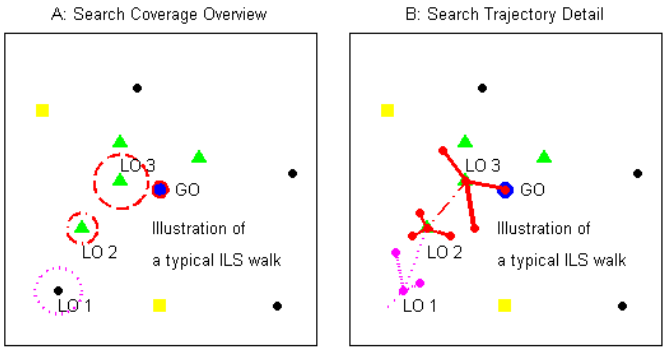


Figure 4.11: Comparison between SCO versus STD Modes

local optima if the new local optima is not accepted by the acceptance criteria of ILS[14]. These details can only be seen if each s^t is assigned to a specific position in VSPACE. For this, we need another layout algorithm.

Formal Definition of Search Positions Layout Problem

This phase aims to place $s^t \in ST, t \in \{0, 1, \ldots, lastItr\}$ on VSPACE while minimizing the layout error $errST$ between s^t and its k-nearest APs, $k \ll |APset|$. Currently, we set $k = 3$ since at least 3 points are required to resolve layout ambiguity. We denote $AP_{nearest(i)} \subset APset$ as the i-th (1st, 2nd, 3rd) nearest AP from s^t. We use $weight(i)$ to control the layout. We want to place s^t as close as possible to the 1st nearest AP, then to 2nd nearest AP, and finally to the 3rd nearest AP. The search position layout error `errST` is computed as follows:

$$errST = \sum_{i=1}^{k=3} weight(i) * abs(ELD(s^t, AP_{nearest(i)}) - d(s^t, AP_{nearest(i)})) \tag{4.4}$$

See Figure 4.12 for illustration. Suppose the three nearest APs to s^t at time t are: AP_x, AP_y, AP_z. The distance of s^t w.r.t these three APs are indicated by the radius of the enclosing circle. If we install s^t in the position 1 shown in Figure 4.12.A, then the `errST` is still quite large (visualized as **green thick lines**). A better alternative is to install s^t in the position 2 shown in Figure 4.12.B with much smaller `errST`.

[14]See Section 7.2 for this ILS behavior on Traveling Salesman Problem (TSP).

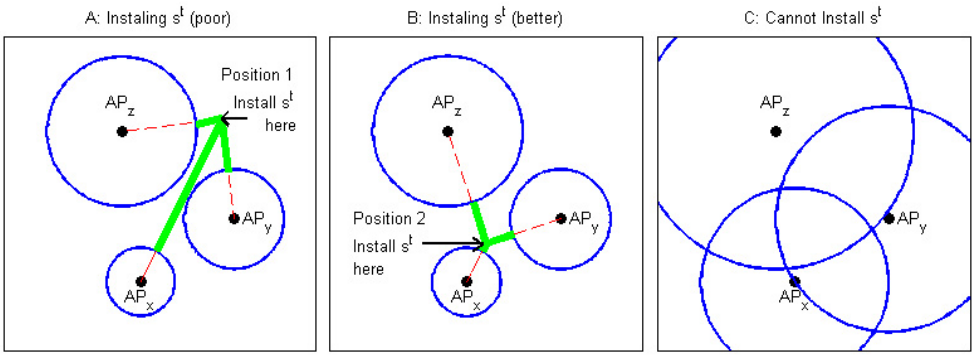


Figure 4.12: Determining the Position of s^t at Time t in VSPACE

Recall that the information from an $AP\ X$ is not meaningful when $d(s^t, AP\ X) > \delta$. We do not draw s^t on VSPACE if $d(s^t, AP_{nearest(1)}) > \delta$ in both SCO and STD mode. Figure 4.12.C shows that when s^t is too far from any known APs (visualized as large enclosing circles), it is hard to decide where to install s^t without causing a large `errST`.

Heuristic for Search Positions Layout Problem

To obtain the layout of each point in ST, we use a slightly modified spring model layout algorithm as follows:

ERRST($s^t, Direction, StepSize, AP_1, AP_2, AP_3$)

```
1  Move s^t by StepSize units according to Direction
2  errSTvalue ← 0
3  for i ← 1 to 3
4      do errSTvalue ← errSTvalue + weight(i) * ABS(ELD(s^t, AP_i) − d(s^t, AP_i))
5  Reverse s^t to original position before this move
6  return errSTvalue
```

SEARCH-POSITIONS-LAYOUT($ST, MaxIteration$)

```
1  for s^t ← s^0 to s^lastItr ∈ ST
2      do Let AP_x, AP_y, AP_z be three nearest APs w.r.t s^t (ascending)
3          if s^t is far from AP_x, AP_y, AP_z
4              then Coordinate(s^t) ←NOT-DRAWN and continue to next s^t
5          Coordinate(s^t) ← Coordinate(AP_x)
6          for Step ← 64 to 1 // StepSize /= 2 at every loop
7              do for i ← 1 to MaxIteration
8                     do BestDirection = Stand-Still
9                        BestLocalSpringTension = ∞
10                       for Dir ← North, East, South to West
11                           do Tension ←ERRST(s^t, Dir, Step, AP_x, AP_y, AP_z)
12                              if Tension < BestLocalSpringTension
13                                  then BestLocalSpringTension ← Tension
14                                       BestDirection ← Dir
15                       Move s^t to BestDirection for StepSize units.
16 return {Coordinate(s^0), Coordinate(s^1), ..., Coordinate(s^lastItr)}
```

Line 2-15 are repeated for each solution in ST. In line 2, we find 3 nearest AP w.r.t s^t in $O(|APset| * n)$ time. Line 3-4 determines whether s^t is too far from known APs or not. Line 5-15 is our 'modified spring model' layout algorithm that initially places s^t at the same coordinate as $AP_{nearest(1)}$, then the heuristic will try to move s^t by 64 units to 4 directions. If such a move reduces the local spring tension between s^t and its 3 nearest APs, s^t is moved there. This heuristic then tries to move s^t by 32, 16, 8, 4, 2, 1 unit(s) until no more moves can reduce the spring tension. This heuristic runs in $O(|ST| * (|APset| * n + C * MaxIteration))$.

This heuristic produces a deterministic layout of points in ST when given the same $APset$. That is, two same points $s^a, s^b \in ST, d(s^a, s^b) = 0$ are drawn in the same coordinate. This is particularly useful to detect solution cycling phenomenon in SLS run as two identical solutions found in separate time during SLS run are shown in the same position in VSPACE.

Comparison with Run Time Distribution (RTD) Analysis

Our SCO and STD modes are richer than the Run Time Distribution (RTD) analysis. In Figure 4.13.A, we observe that the SLS algorithm experiences stagnation as solution probability does not increase after some time. However, RTD analysis cannot explain the details of the SLS run. In Figure 4.13.B and 4.13.C, we can observe where this SLS algorithm is stuck and are in a better position to design strategies to address this issue.

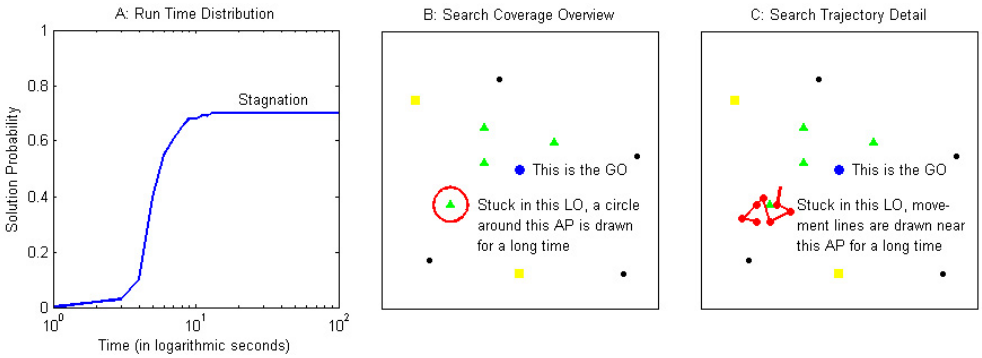


Figure 4.13: Comparison between RTD versus SCO and STD Modes

With these capabilities, SCO and STD modes of FLST visualization can help in answering the questions posed in Section 4.3.

4.6 Multi Instances Analysis

After explaining SLS algorithm behavior on one COP instance, it is natural to extend our questions in Section 4.2 and 4.3 to multi instances to avoid over-fitting and to obtain more sound conclusions. However, as FLST visualization involves working with human user, we cannot use too large training instances. In general, we want to further ask these important questions:

1. Are these fitness landscape characteristics (Section 4.2) found to be general across all COP instances?
e.g. if instance A has a Big Valley structure, does instance B have it too?

2. Are the observations of an SLS algorithm behavior on one individual COP instance (Section 4.3) generalize-able to other instances?
e.g. if the SLS runs well on instance A, does it runs well on instance B too?

The possible answers for each question are:

1. Yes.
All these COP instances exhibits the same fitness landscape characteristics.
The observations seem consistent throughout many (class of) instances.
Perhaps there is a generic property which should be further tested.

2. No.
Can similar instances be grouped into classes of instances?
Perhaps a different customized SLS algorithm is needed?

To answer these two questions, we simply apply the analysis using FLST visualization to the entire training instances as shown later in Chapter 7.

4.7 Summary

1. When one wants to understand the behavior of an SLS algorithm on a COP fitness landscapes (in order to address the SLS DTP), there are *many* questions that need to be answered (see Section 4.2 and 4.3). There are several white-box approaches for explaining SLS algorithm behavior, but these approaches still have difficulties in answering questions about SLS algorithm behavior (see Section 4.4).

2. We propose an idea to visualize the *approximate* fitness landscape and search trajectory on it (called the FLST visualization). This visualization is not meant to replace the existing white-box approaches but rather to augment them. The key concepts of this FLST visualization are:

   (a) We select some diverse and high quality local optima solutions in the fitness landscape (using the SLS algorithm itself) – called *anchor points* (*AP*s).

   (b) We layout these *AP*s on VSPACE with a spring model layout algorithm that utilizes pairwise distance between each *AP*. This is to bring down the n-dimensional combinatorial solution space to a more user friendly 2-dimensional VSPACE.

   (c) We label these *AP*s according to their solution quality to make up the fitness landscape visualization.

   (d) Then, points along the search trajectory are drawn (laid out) w.r.t these *AP*s in an animated fashion in order to explain the SLS algorithm behavior.

3. FLST visualization is our attempt at visualizing a complex n-dimensional fitness landscape in a simpler 2-D visualization. In order to do that, we essentially introduce visualization errors. Here, we reflect on the current limitations that can be improved in the future.

   (a) In Section 4.5.2, we show that in order to build the FLST visualization, we collect a fixed and small amount of Anchor Points *AP*s using the SLS

algorithms themselves. This limits the FLST visualization to show only a *fraction* of the *explored* fitness landscape.

(b) In Section 4.5.3, we show that we cannot layout the n-dimensional APs in a perfect way. We quantify the layout error with $errAP$ and show that COP with lower/higher $DiversityIndex$ has lower/higher $errAP$, respectively. With higher $errAP$, the user must be more cautious on what he observes, as shown in the examples in Chapter 7 later.

Later in Section 7.2, we show that when the COP has low $DiversityIndex$ and the AP layout has low $errAP$, like in TSP, the users can expect to see the search trajectory ST as a rather smooth movement between APs.

Later in Section 7.3 and 7.4, we show that when the COP has medium to high $DiversityIndex$ and the AP layout error is substantial, like in QAP and LABSP, the users can only see the search trajectory ST as 'fragments': the ST appears close to one AP, disappears for some iterations (the 'blank period'), and then reappears in another AP that is usually far from the previous AP.

(c) In Section 4.5.4, we show two ways to display the position of the search trajectory w.r.t the APs that have been laid out. The Search Coverage Overview (SCO) mode is safer but less accurate. The Search Trajectory Detail (STD) mode is more accurate but has to introduce another layout error $errST$. But the two modes cannot show the search trajectory when it is currently far from any of the collected APs (the 'blank period').

4. FLST visualization is generic as it only uses generic properties, e.g. objective values, distance information, and time. It does not depend on the COP or SLS algorithm used, making it suitable to be used for addressing the SLS DTP.

5. The interpretation of FLST visualization depends on the search strategies being used, problem specific knowledge, fitness landscape structures, etc. The interpretation from one single training instance is then tested on other training instances (different fitness landscapes) to check for robustness.

4.8 Looking Ahead

In the next chapter, we present our SLS visualization tool Viz, which implements the FLST visualization and much more.

In Section 8.2, we present future works to improve this FLST visualization.

Chapter 5

SLS Visualization Tool: Viz

If one picture is worth ten thousand words ...
Then a good animation & interactive user interface can be worth many more still ...
— **Modified Chinese Proverbs**

In this chapter, we elaborate the other white-box SLS visualizations and UI aspects in our visualization tool: Viz. *Parts of this chapter have been published in [60].*

5.1 Overview

The novel Fitness Landscape Search Trajectory (FLST) visualization described in Chapter 4 is a rather complex visualization. For it to be useful and user-friendly, it needs a UI and dedicated tool to generate and then playback the visualization/animation from data collected via SLS run(s). We have provided such a tool, an SLS engineering suite named as Viz. In this chapter, we focus on the SLS visualization aspects of the tool, Viz SIMRA (**S**ingle **I**nstance **M**ultiple **R**uns **A**nalyzer) shown in Figure 5.1.

As seen in Figure 5.1, there are more visualizations than just the *generic* FLST visualization (label A) in SIMRA. They are Objective Value (OV) visualization (label B) to show OV information over time, Fitness Distance Correlation (FDC) visualization (label C) to analyze the fitness landscape, and Event Bar visualization (label D) to highlight generic events occurring during the search. Generic visualization is a powerful concept because it is independent[1] from the underlying SLS algorithm and COP. These generic visualizations are the strengths of Viz SIMRA.

Other than these four generic visualizations, Viz SIMRA also has Algorithm-Specific (AS) (label E) and Problem-Specific (PS) (label F) visualizations. These AS and PS visualizations are linked with the SLS algorithm being used and the COP being solved, respectively.

[1] This is possible in the context of SLS algorithm for COP because every COP has a fitness landscape model [74, 89, 118, 68] and every SLS algorithm works by (locally) mutating the current solution along this fitness landscape with respect to generic properties such as OV, distance, time (iterations).

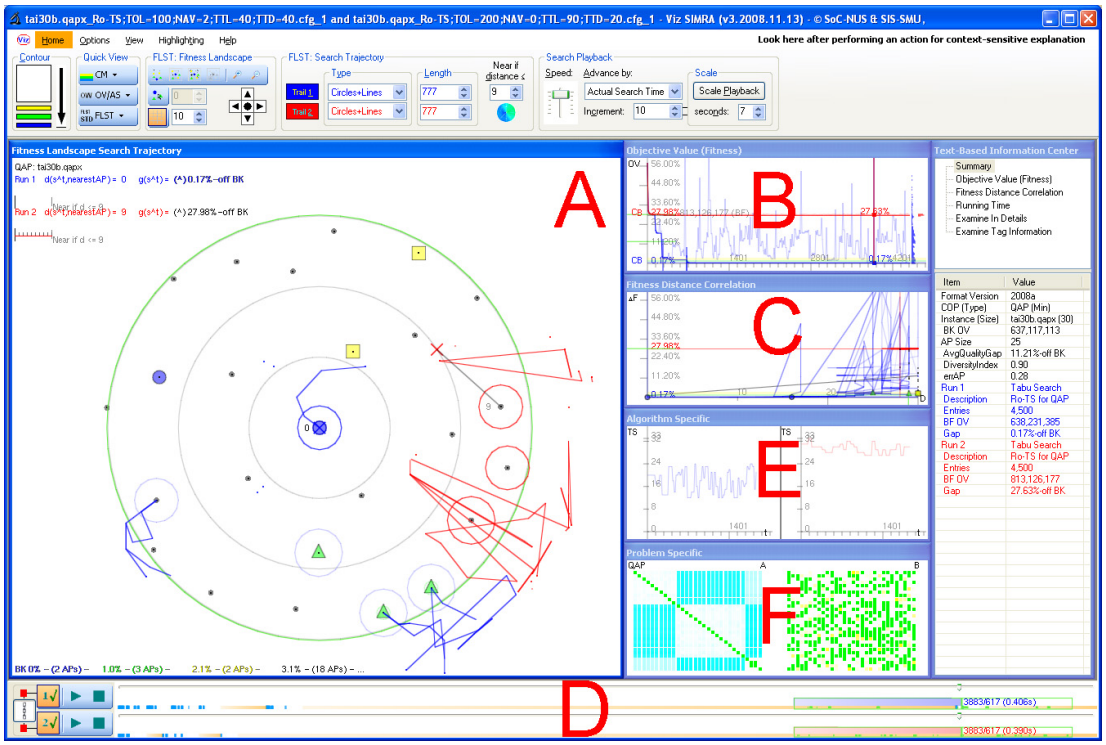


Figure 5.1: Viz v3.2008.11.13: Single Instance Multiple Runs Analyzer (SIMRA)

These other generic, algorithm-specific, and problem-specific visualizations are generated from the same SLS run(s) data as with FLST visualization (i.e. all are off-line visualizations). Section 5.2 shows the design choices of each visualization and how each of them can complement FLST visualization in analyzing SLS behavior[2].

Then, in section 5.3, we further elaborate the user interface aspects of Viz tool that we have designed to maximize the strengths of these visualizations.

5.2 Visualizing SLS Behavior in a Holistic Manner

In this section, we elaborate the design choices of the SLS visualizations that we use on top of FLST visualization. They help the analysis in the Chapter 7 experiments.

5.2.1 Objective Value (OV) Visualization

Objective Value (OV) a.k.a fitness, is often the key attribute that drives the SLS algorithm. Typical OV visualization plots a time series of OV (y-axis) over iteration/time (x-axis) as partially shown as the **green line** in Figure 5.2 label **A**.

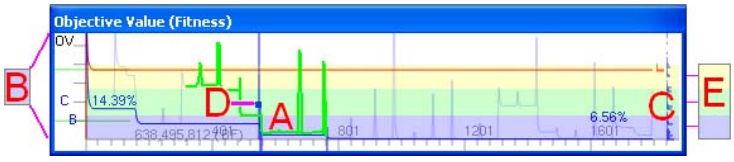


Figure 5.2: Objective Value over Time

[2]These visualizations follow the *information visualization* guidelines [144, 145, 146, 152, 147].

In Viz, we enhance this OV visualization with additional information to help the user to understand the overall context of how the OV is changing over time:

1. Scale of y-axis (see Figure 5.2 label **B**): In a minimizing COP, we set Min = Best Known BK OV (or if it is unknown, Best Found BF OV) and $Max = (100 + MaxDeltaFitness)\% * Min$. $MaxDeltaFitness$ is customize-able. This forms the scale of y-axis. The y-axis labels can be shown as percentage-off (e.g. 16680 is 10%-off if the BK OV is 14379) or as absolute value. Min & Max roles are reversed in a maximizing COP.
2. Frequency histogram drawn vertically along y-axis in *logarithmic* scale (see Figure 5.2 label **C**). This histogram highlights the OV distribution of the solutions found by the SLS run. The average OV throughout the SLS run is also highlighted.
3. A line to indicate best-found-so-far and percentage-off indicator w.r.t BK OV.

We purposely draw the OV of the current solution in the 'middle' (see Figure 5.2 label **D**) of the OV fluctuation line[3] so that the visualization is split into 'immediate past' and 'immediate future'. The y-position of an OV is computed with respect to the BK OV with[4] the formula $\frac{abs(OV-BK)}{BK} * 100.0\%$. BK OV is supplied by the user (e.g. from benchmark instances). The OV of the current solution is *animated* as the SLS run is being played back. This design choice tells the user the quality of the current solution within the context of its immediate past and future[5].

We use the same contour map colors used in the FLST visualization (see Section 4.5.3) on the background (see Figure 5.2 label **E**). Thus, the user can quickly draw connections between the current search trajectory and its solution quality.

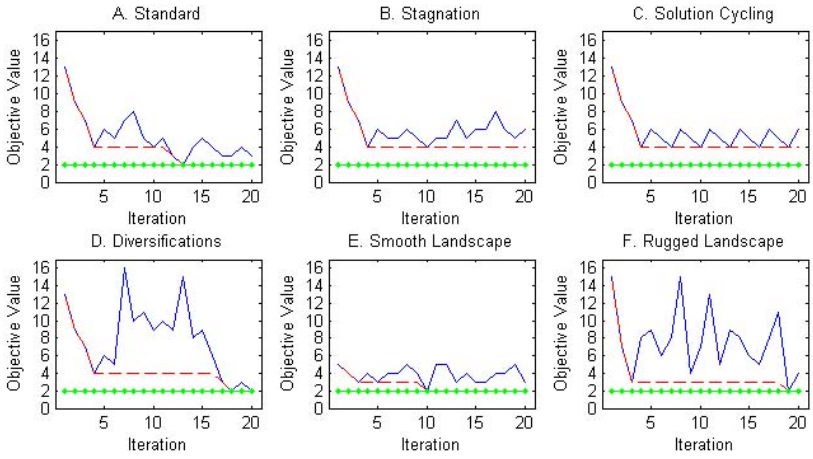


Figure 5.3: Potential Structures seen in the OV Visualization

[3]This is also known as 'sparkline' according to Tufte [147].

[4]This formula suffers from a singularity when BK OV is 0, i.e. $\frac{abs(y-0)}{0}$ is undefined $\forall y$. This formula can also produce relatively larger percentage-off when BK OV is a small value near 0 compared if BK OV is larger for the same deviation value, e.g. $\frac{abs(11-10)}{10}$ is 10%-off from BK OV whereas $\frac{abs(101-100)}{100}$ is just 1%-off from BK OV, although both cases have the same deviation value = 1.

[5]Recall that an on-line OV visualization can only have past and current data! This also complements FLST visualization as FLST visualization is not designed to show SLS behavior from this angle.

There are common patterns frequently observed in the OV visualization. We listed some examples (not exhaustive) in Figure 5.3 – assuming a minimizing COP:
1. Standard case: the SLS run quality gradually improves over time.
2. Typical stagnation: after certain time, the quality does not improve anymore.
3. A typical solution cycling pattern: obvious repetitions.
4. There will be 'spikes' if strong diversifications are called every certain interval.
5-6. If the fitness landscape if quite smooth, the OV fluctuation between adjacent solutions is not big. It is the other way around if rugged.

5.2.2 Fitness Distance Correlation (FDC) Visualization

FDC analysis (see Section 3.5.2) is used to give a *rough measure* of the COP's difficulty. We visualize the FDC information as a scatter plot (see Figure 5.4 label **A**). In Viz, we plot the fitness difference of each AP with the Best Found BF (AP_0) along the y-axis and distance between each AP with the *nearest* $BF \in APset$ where $g(nearest\ BF) == g(AP_0)$ along the x-axis. This scatter plot quickly shows whether there exists a correlation, be it a positive or negative one, and whether such correlation is strong. A simple linear regression line is added to highlight the trend.

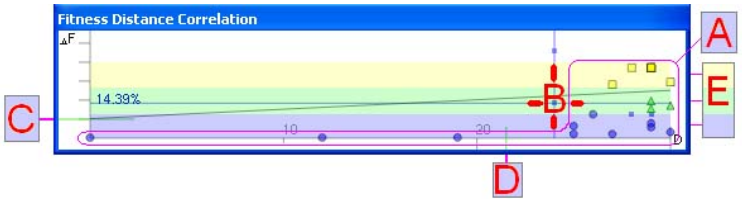


Figure 5.4: Fitness Distance Correlation

Unlike Objective Value (OV) visualization that uses Best Known BK OV as baseline, the FDC scatter plot in Viz uses the Best Found BF solution[6] as baseline since the SLS algorithm may *not* actually reach the BK solution. In the OV visualization, using the BK OV is fine as we just need to compare OV. However, the FDC scatter plot requires not just the OV difference, but also distance information computed by comparing solutions. As we need to get the actual baseline solution - not just its OV, we can only use BF solution(s) that are really found by the SLS algorithm.

In Viz, we show more information in the FDC scatter plot. We add an animation of the position of the current solution w.r.t the nearest BF by plotting the fitness-distance (F-D) information over time using a cross-hair highlight. This is to quickly gauge how good/bad and how near/far the current solution is w.r.t the nearest BF as shown by the cross-hair position (see Figure 5.4 label **B**). This FDC visualization also has the same contour map (see Figure 5.4 label **E**) as in the FLST and OV visualization.

While the y-axis (delta fitness) is adjustable based on the percentage-off w.r.t the BF OV (as in the OV visualization), the maximum x-axis value is strictly equal to the maximum distance, that is, the size of the COP instance n. This arrangement is useful to gauge the distribution of solution quality (see Figure 5.4 label **C**) and distances (see Figure 5.4 label **D**) of local optima w.r.t the nearest BF.

[6]Note that this may lead to erroneous conclusions if these BF solutions are far from true GO.

FDC analysis has limitations, but it is good to quickly portray some fitness landscape properties. It is conjectured that for a minimizing COP, the COP will likely have one of the following r_{FDC} values and scatter plots (see Figure 5.5):

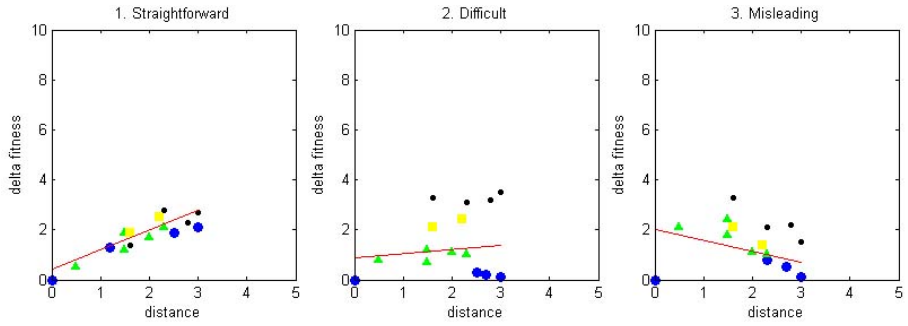


Figure 5.5: Potential Structures seen in FDC Visualization

1. 'Straightforward' (a.k.a the 'Big Valley'), when r_{FDC} is approaching +1.0.
   - ∗. Fitness increases as the SLS trajectory is approaching global optima.
   - ∗. Visualized as a linear regression line.
   - ∗. This is a relatively easy COP.
2. 'Difficult', when r_{FDC} is near 0.0.
   - ∗. Low or no correlation between fitness and distance w.r.t global optima.
   - ∗. Visualized as a regression line that is almost parallel to x-axis.
   - ∗. There exist good local optima far from the global optima.
3. 'Misleading', when r_{FDC} is approaching −1.0.
   - ∗. Fitness decrease as the SLS trajectory is approaching global optima.
   - ∗. Visualized as a linear regression line with negative slope.
   - ∗. This can be a frustrating COP for an SLS algorithm.

5.2.3 Event Bar Visualization

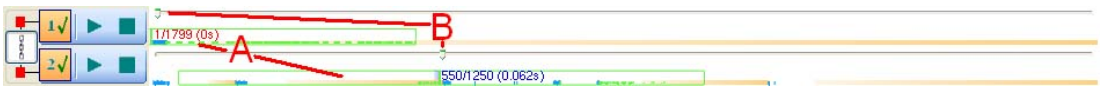


Figure 5.6: Event Bar and the Iteration Slider

Viz SIMRA allows the user to decide which part of the (off-line) search playback should be visualized by clicking and dragging the time/iteration slider (see Figure 5.6 label **B**). Since the SLS algorithm usually runs for a large number of iterations, it might be too painful for the user to scan the entire search playback in FLST visualization. To highlight the interesting portions, which may be missed or not obvious as they are rare, the Event Bar visualization highlights these 'index points':

- 'New Best Found' – **blue bars** and 'Series of Non Improving Moves' – **shades of orange**. 'New Best Found' occurs at iteration t if at iteration t, the search found

a solution which has OV better than iteration [0 . . . t-1]. In between the 'New Best Found' events, we draw some 'Series of Non Improving Moves' highlights to show when the search is experiencing stagnation.

- 'Near set of Anchor Points' – **green bars**. Occurs on iterations where the current solution is near to at least an $AP \in APset$ (distance $< \delta$ units set by user).

- Information about current (recent) iterations: iterations elapsed/left, actual search time, and highlight parts of the search trajectory currently shown on the screens.

This Event Bar visualization is drawn below the time/iteration slider (see Figure 5.6 label **A**). The information gained from Event Bar visualization can assist the user to quickly move around in time during the search playback.

5.2.4 Algorithm-Specific (AS) Visualization

Algorithm-Specific (AS) visualization displays the change of the SLS *dynamic* parameters over time. This may help explain the SLS behavior over various time points that is not shown in the FLST visualization. Typically, the information from this AS visualization should be related with the information from other visualization(s) to be more meaningful.

For this thesis, we only support AS visualization of the (possibly) dynamic part(s) of: Tabu Tenure (TT) over time for Tabu Search (TS) and perturbation strength plus the acceptance/rejection status for Iterated Local Search (ILS).

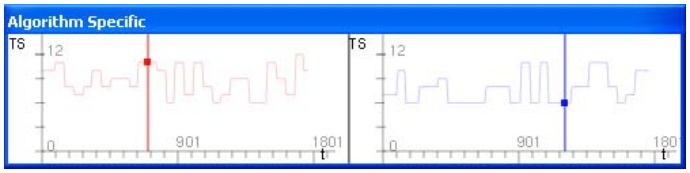


Figure 5.7: Algorithm-Specific Visualization for TS

An example of AS visualization for TS is shown in Figure 5.7. TT information recorded in the log files is presented as a time series visualization. For Reactive [15] or Robust TS [137], TT dynamically changes throughout the search. We can use this AS visualization to explain – for example – why TS trajectory is more diverse (during high TT phase) or more focused (during low TT phase).

5.2.5 Problem-Specific (PS) Visualization

Problem-Specific (PS) visualization[7] is an intuitive visualization as it is directly related to the COP being solved. There is information that can be gained by observing the PS visualization. We can get a glimpse of the problem structure which can be exploited for better performance, e.g. clustered versus distributed cities in Traveling Salesman

[7]Some SLS tools have PS visualizations, e.g. COMET [149] has a built-in interface that allows user to implement PS visualizations. Human-Guided Search [79] also relies a lot on PS visualizations.

Problem (TSP). We can also verify the correctness of our problem specific search strategy, e.g. if we want to keep short edges in the TSP tour; does our SLS algorithm really keep short edges? This is not shown in the FLST visualization.

For this thesis, we only support three PS visualizations: TSP, Quadratic Assignment Problem (QAP), and Low Autocorrelation Binary Sequence (LABS).

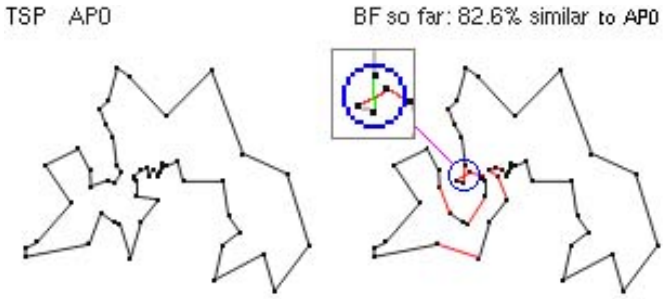


Figure 5.8: Problem-Specific Visualization for TSP (berlin52)

An example of PS visualization for TSP[8] solutions/tours is shown in Figure 5.8. Here, we see TSP tours for TSP instance 'berlin52' from TSPLIB [143, 119]. On the left is an optimal tour and the right is an example of a non-optimal tour as there is an obvious crossing in the tour (see the small **blue circle**). Visualizing TSP tours like this enables a human to compare the differences between the current tour with a baseline (usually good) TSP tour and to quickly spot crossings – a 'bad feature' in a TSP tour.

PS visualizations have their limitations. While some COPs have natural visualizations, particularly those which can be cast in a spatial setting (e.g. TSP), it is not clear how to do it in (most) other cases. There is also information overload since looking at the full solution has too much detail. It is also harder to visualize the search trajectory since PS visualization focuses too much on the current solution in gory detail but does not show what is going on in the SLS algorithm across a time interval. For example, it might be difficult to see if an SLS algorithm for TSP is trapped in a local minimum by looking at an animation of consecutive TSP tours found by the SLS algorithm. It is hard to verify whether the current tour indeed has been shown X iterations ago.

5.3 User Interface Aspects

The presentation of the visualizations in Viz SIMRA are further enhanced by using the following user interface features.

5.3.1 Coordinated Multi-Source Visualizations

Each visualization described in Section 4.5 and 5.2 is capable of explicating *some* aspect of COP fitness landscape characteristics and/or SLS trajectory behavior on the fitness landscape. However, relying on a single visualization *alone* can be myopic and the full picture of what is happening during the search may be unclear. We believe that

[8]The early works for visualizing TSP tours, especially to assist man-machine interactive optimization, begun in 1960s [92, 81]. A recent work is Human-Guided Search in Section 3.5.2.

multiple coordinated points of views[9] (i.e. show the same data at time t from various angles) are required given the difficulty of analyzing SLS behavior. Some scenarios:

1. We can use the Event Bar highlights to navigate to iterations where interesting events occurred, e.g. 'New Best Found' event, observe the FLST visualization to see the details, and then verify such an improving step in the OV visualization.

2. We observe a 'solution cycling' phenomenon in the FLST visualization and realize that the cause is low tabu tenure as observed in the Tabu Search (TS) Algorithm Specific (AS) visualization.

3. We observe a 'Big Valley' pattern in the FLST visualization and verify it in the FDC visualization. Then, we check when the SLS trajectory is far from the middle of the FLST visualization, it also has poor OV in the OV visualization.

Situation Awareness theory in Psychology [41] reminds us that the human is unable to observe multiple visualizations (displays) at the same time if they all require high attention. When the human is bombarded with a number of displays, his overall scanning ability drops, making him concentrate on only a small fraction of the displays, thus losing Situation Awareness.

To reduce this issue in Viz, we designate the FLST visualization as the *main* visualization – the focus of user's attention; the other visualizations are peripheral. Nevertheless, as the search playback in Viz is *not* on-line, this issue is not severe as the user can always pause, rewind, or replay the search if some information was missed.

Viz presents these coordinated visualization windows in a Multiple Documents Interface (MDI) style. This conforms with the Situation Awareness theory in reducing information overload as user has the freedom to show, hide, scale, or layout the visualization child window(s). The user can concentrate on FLST visualization and recall peripheral window(s) only when needed.

5.3.2 Visual Comparison

The human is better at relative than absolute discrimination[10]. Viz SIMRA exploits this fact and has a capability to visually compare the SLS behavior (see some comparison modes in question 10 of Section 4.3). SIMRA allows the user to load two different SLS runs on the same COP instance (or even the same SLS run twice) to run 1 and run 2 slots in SIMRA GUI. These two SLS runs can be played back concurrently.

Visualizations of both runs can be drawn in either juxtaposition (side-by-side), e.g. the quality of two TSP tours are easily compared in Figure 5.8; or superimposition (overlap), e.g. the difference of the **red** (stuck) and **blue** (reach BF) SLS runs on the same fitness landscape are clearly shown in Figure 5.1 label A. This feature gives this visualization tool its name: Single Instance *Multiple Runs* Analyzer.

[9] We may be able to cramp all individual visualizations into a 'big' visualization but it will be very convoluted and thus ineffective.

[10] A simple illustration: The question "Is a soccer ball bigger than a golf ball?" is easier to be answered relatively rather than precisely measure the volume of both soccer ball and golf ball and then answer the question in absolute manner.

5.3.3 Animated Search Playback

As the SLS algorithm typically runs for a large number of iterations, the most natural way to visualize its search trajectory is to playback the search dynamically over time with animation. Information accumulated over time is hard to be seen in a static fashion as drawing everything in one screen tends to clutter the visualization and hide details. The static version is more suitable to show the search coverage. Figure 5.9 shows a comparison of static (overview) versus animated (detailed) visualization. In the animated version, we can see that the SLS algorithm spends substantial time (0.296-0.765s) on the same region (stuck). This is not clearly shown in the static version.

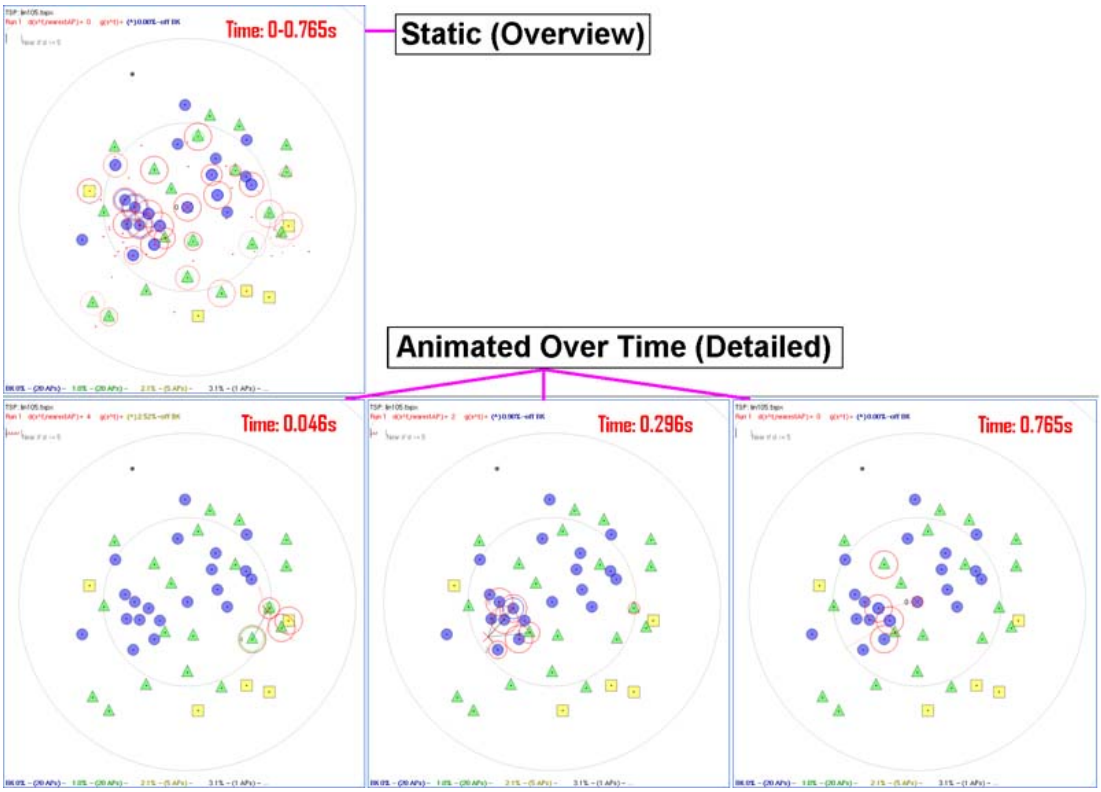


Figure 5.9: Static versus Animated Presentation

In Viz, the user is allowed to specify the search trajectory animation length l to show a trail of consecutive points $s^{t-l}, s^{t-l+1}, \ldots, s^t$ on various SLS visualizations at time t. By changing l, the user can adjust the tradeoff between overview and detailed displays. Essentially, $l = t$ (all points up to time t) in overview mode.

The animation effect is achieved by drawing series of consecutive visualizations with small playback time increment. For smooth transitions, Viz uses *weighted alpha blending*: gradually fading out the color of a trail's tail. This way, the user knows that the darker colored lines/circles on the trail are the current ones.

Viz also allows the user to adjust the search playback speed[11], which determines how fast the animation will be drawn. This is essential, as different individuals have different visual capacities in discerning the information from the animation.

[11] This is do-able as Viz is an offline visualization tool. Playback speed != true SLS algorithm speed.

Viz supports two search playback modes: based on 'iteration count' or based on (slowed down) 'running time'. This is because different SLS runs with the same iteration count may take different running time and playback based on 'running time' can be more fair in this case. On the other hand, playback based on 'iteration count' can be used to observe the changes done per iteration.

5.3.4 Color and Highlighting

Given the complexity of the visualizations in Viz, it is important to assist the user to quickly find what he wants. Highlighting – making certain things *clearly* stand out from the rest – is a helpful visual aid to address this issue.

Highlighting is possible because human has a pre-attentive visual processing system [152] where some visual features are distinguishable *before* conscious attention, e.g. colors, grey-scale levels, sizes, shapes, textures, orientations, labels, etc. Making an object significantly different from its surroundings on at least one pre-attentive dimension ensures that it can be detected by a viewer effortlessly and at high speed.

Different forms of highlights have different effectiveness in different context, e.g. it is hard to differentiate object's color when there are many colors, or to differentiate object's size when there are many similar-sized objects. In Viz, we decide to avoid having too many colors (e.g. see Figure 4.6), thus we can mainly use color for highlighting.

Other than for highlighting, color is also good for coding, labeling, or categorizing. If the same color is used for different but related objects, the human user will process them as being associated even when they are drawn in separate screens.

Some scenarios of the usage of color and highlighting features are:

1. In the FLST, OV, and FDC visualizations, color+shape labels are used to form a contour map that shows the solution quality attribute. Changing the map value ranges can be used to quickly differentiate solution qualities (see Figure 4.6).

2. In the Event Bar visualization, we draw stripes with different colors to highlight different generic events throughout the search playback (see Figure 5.6).

3. In the FLST visualization 'highlight mode', the user can point to a specific AP. Then, line highlights with different colors will appear to indicate other APs that are too close, average, or too far from the selected AP (see Figure 5.10, right).

5.3.5 Multiple Levels of Details

Considering the limitation of the human visual system when overwhelmed with data, information visualization puts the emphasis on presenting the overview first, gives the human capability to zoom-in into the relevant data, filter the irrelevant ones, and see the details on demand. In order to do this, the visualization system should be able to present the *same* data at multiple Levels of Details (LoD). Essentially, the system will draw more details (highlighting the important ones) when zoomed-in and draw less (draw a summary, hiding the non-important ones) while zoomed-out. Some scenarios:

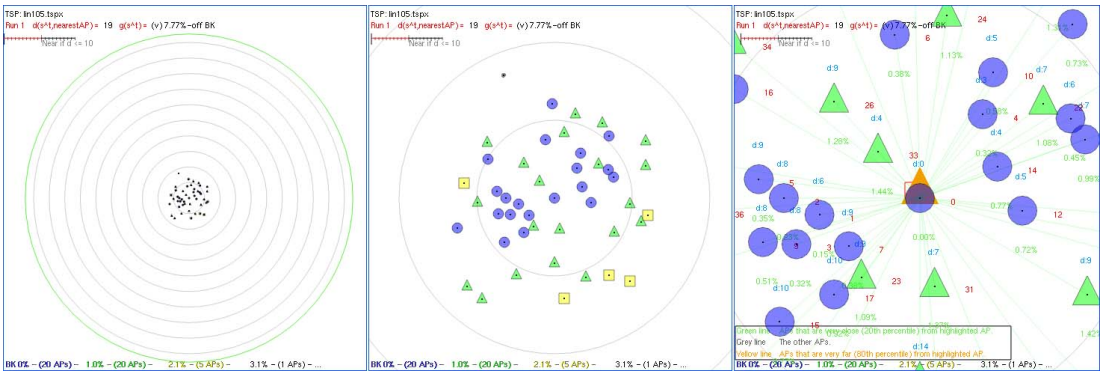


Figure 5.10: Levels of Details Feature in FLST Visualization

1. The time series chart: the OV and AS visualizations have an x-axis scaling feature so that the entire run can be visualized in one screen if needed (see Figure 5.2).

2. Zooming-in and out in the FLST visualization can reveal information about the COP fitness landscape. In Figure 5.10, we have the FLST visualization in zoomed out mode (left), normal view (middle), and zoomed in mode (right). Here we observe the Big Valley property of TSP (elaborated more in Section 7.2).

3. We draw more data when the visualization window is enlarged and vice versa. In the OV, FDC, and AS visualizations, the labels along the x-axis and y-axis scale are shown if the window size is big enough, and hidden otherwise.

5.3.6 Text-Based Information Center (TBIC)

Some information is still best displayed as text[12], e.g. as a list of data. But we cannot add too much text in the visualization as it will clutter the screen, especially for visualization with small screen space. Some detailed information must be displayed as text *outside* the visualization window. This is the main purpose of the Text-Based Information Center (TBIC) window (see Figure 5.11).

TBIC window can also include statistics to support the visualization counterparts, e.g. the average OV is drawn as a stripe along the y-axis of the OV visualization, but the actual numeric value is also in the TBIC window. To further save screen space, we do not display all textual information but let the content in the TBIC window be context-sensitive according to which visualization window is being activated by the user. The textual information available in TBIC window are:

Summary: When the FLST visualization is activated, a summary information about the COP fitness landscape (AverageQualityGap, DiversityIndex, and `errAP` are shown, see Section 4.5.2-4.5.3) and information about the SLS run are given.

[12]There is an SLS algorithm analysis tool like Viz that is designed without visualization, e.g. EasyAnalyzer by Di Gaspero *et al.* [33]. EasyAnalyzer implements *existing* white and black box tools, e.g. basin of attraction analysis (for analyzing COP search space), RTD analysis (for analyzing SLS run time behavior), and F-Race (black-box tuning algorithm) and show the results as text.

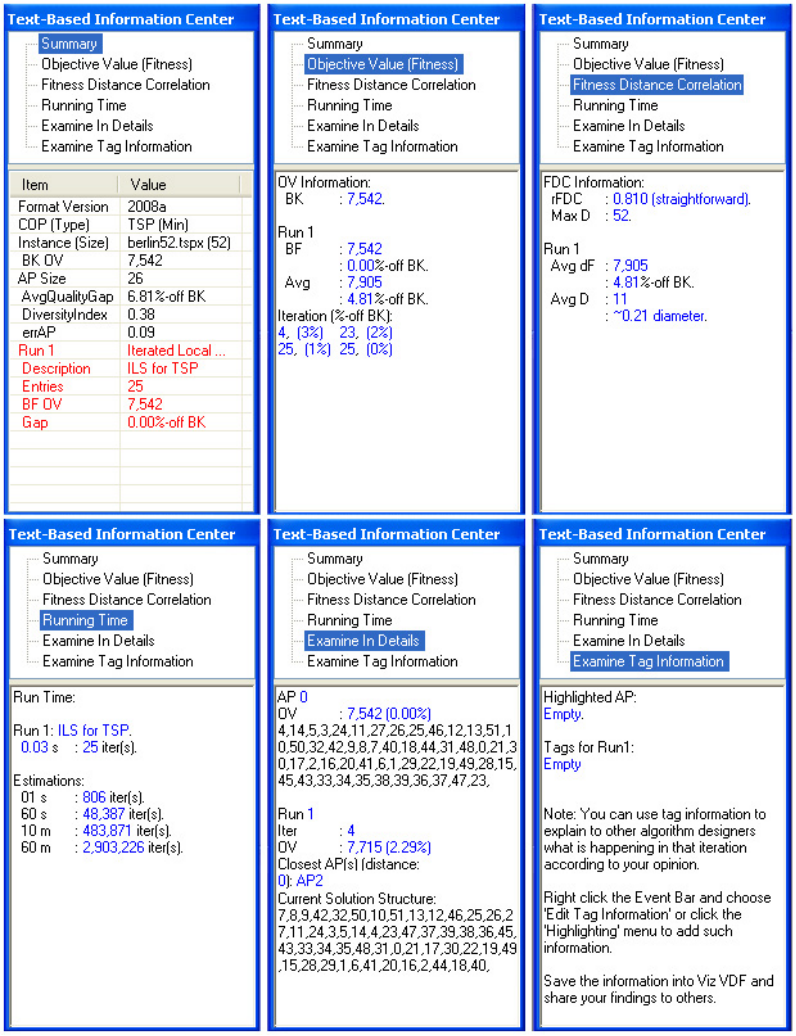


Figure 5.11: TBIC: Summary, OV, FDC, Running Time, Details, Tags

Objective Value (Fitness): A statistical summary of OV: BK, BF, average, etc, are displayed. In visual comparison mode, the qualities of run 1 and 2 are compared.

Fitness Distance Correlation: A statistical summary of FDC: r_{FDC}, maximum distance, average delta fitness, and distance w.r.t BF, etc, are displayed. In visual comparison mode, the average delta fitness and distance of run 1 and 2 are compared.

Running Time: An estimation of run time for longer runs is displayed. This information is projected from the actual search time information recorded in the RunLogs. This is to help the algorithm designer to plan the execution time for his algorithm. In visual comparison mode, the run time of run 1 and 2 are compared.

Examine In Details: The user can view the detailed information about the currently highlighted anchor point and the information about the current solution at that time. The user can point to a specific AP in the FLST visualization to reveal its details (OV, the solution structure, etc).

Examine Tag Information: The user can view the list of tag information.

5.4 Summary

1. Viz SIMRA is an SLS visualization tool that displays the FLST visualization (see Chapter 4) and other supporting white-box SLS visualizations (OV, FDC, Event Bar, AS, PS) as user-friendly as possible.

2. To further enhance the presentation, we develop the following user interface aspects: coordinated multi-source visualizations, visual comparison, animated search playback, color and highlighting, multiple levels of details, and text-based information center.

5.5 Looking Ahead

With this SLS visualization tool Viz which implements the novel FLST visualization, is the SLS Design and Tuning Problem solved?

The answer depends on how these visualizations are used. FLST and the other SLS visualizations are white-box analysis tools which requires a human expert to analyze, interpret, and take actions based on the analysis results. If not used properly, these visualizations cannot address the SLS DTP.

In the next chapter, we present the Integrated White+Black Box Approach, which combines the correct usage of SLS visualizations with black-box tuning algorithms. This combination is our proposed solution to address the SLS DTP.

For more details about Viz, please consult the Viz website:
`http://www.comp.nus.edu.sg/~stevenha/viz`.
Alternative URL:
`http://sls.visualization.googlepages.com`.

Chapter 6

Integrated White+Black Box Approach

Two are better than one, because they have a good return for their work.
If one falls down, his friend can help him up.
But pity the man who falls and has no one to help him up.
— **Ecclesiastes 4:9-10 [159]**

In this chapter, we discuss human+computer collaboration for addressing the SLS DTP in a better way. We combine both white-box (which use human strengths) and black-box approaches (which use computer strengths). A more complete view of the SLS engineering tool Viz *beyond SLS visualization is also presented. The essence of this chapter has been published in [61].*

6.1 Motivation

6.1.1 White-Box SLS Visualization: Pro and Cons

In Chapter 4, we presented the Fitness Landscape Search Trajectory (FLST) visualization to analyze the COP fitness landscape and the SLS trajectories on it. Then in Chapter 5, we have presented the SLS visualization tool Viz that implements this FLST and some other white-box visualizations (OV, FDC, Event Bar, AS, PS).

FLST visualization, perhaps augmented by other SLS visualizations in Viz, can be used to *understand* the COP fitness landscape characteristics. This helps the algorithm designer to *predict* which search strategies will likely work well on the COP fitness landscape. This prediction can then be *verified* via FLST visualization to see whether the SLS algorithm encounters any 'problem(s)'. These observations can inspire insights (which may be 'outside the box') that are essential for making *informed changes* towards the SLS algorithm design and to *narrow down (focus)* the configuration set. It is hard to arrive at these decisions without effective white-box approaches.

White-box approaches leverage on *human strengths* to analyze and learn from visualizations. Although this is subjective, we show later in Chapter 7 that this process is intuitive and can gain fruitful insights.

However, a well designed SLS algorithm still has configurable parts that may affect the apparent performance. Usually, the configuration set size is still too large to be exhaustively tried manually, necessitating the usage of black-box tuning algorithms.

6.1.2 Black-Box Tuning Algorithm: Pro and Cons

In Section 3.5.1, we have reviewed various black-box tuning algorithms.

Ideally, given a working SLS implementation and an initial SLS configuration set, black-box tuning algorithms can be used to systematically find the most suitable configuration in the given configuration set for solving the COP at hand. This *automated* fine-tuning process is *computer's forte*.

In practice, the size of the configuration set may be huge. Thus, if the algorithm designer wants to have an effective fine-tuning, he must give a 'sufficiently narrow (focused)' configuration set for the black-box tuning algorithm to work with. Furthermore, if the best configuration happens to be outside the initial configuration set, then it cannot be found by fine-tuning alone.

Another issue is that black-box tuning algorithms assume that the given SLS algorithm is designed correctly. If the resulting performance after fine-tuning is still poor, the algorithm designer still has to find what is wrong in the SLS algorithm by himself as these black-box tuning algorithms will not help in discovering the problem.

6.2 The Integrated White+Black Box Approach

A collaboration between two participants will be beneficial if:
1). the respective participant has some advantages that the other has not,
2). the respective advantages are complementary for achieving the common goals, and
3). appropriate interfaces are used.

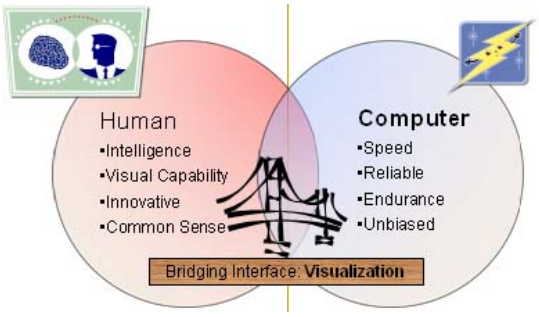


Figure 6.1: Human and Computer Tasks in IWBBA

Figure 6.1 shows the strengths of human[1] and computer. Collaboration between them to achieve a common goal (addressing the SLS Design and Tuning Problem (DTP)) is possible as their unique strengths are *complementary* with visualization as their interface. In this section, we show how white-box approaches, which heavily use human strengths, and black-box approaches, which use computer strengths, are naturally combined to form an Integrated White+Black Box Approach (IWBBA).

[1]For a more detailed elaboration of human strengths, see Appendix C.

Human Tasks (White-box)	Computer Tasks (Mostly Black-box)
Let computer run the SLS algorithm. ⇒	**A. Run 'the box'** (the SLS algorithm).
C. Understand 'the box' (COP fitness landscape and SLS Trajectory behavior).	**B. Visualize information:** Setup FLST visualization, compute statistics.
D. Think outside 'the box': Get insights about the COP + SLS behavior.	⇐ *Wait for human input.*
E. Improve SLS design: add strategies & **narrow down configuration set**	**F. Fine Tune:** Automatically tune the focused configuration space.

Table 6.1: Separation of Human-Computer Tasks in IWBBA

The separation and sequence of tasks in IWBBA that highlights human and computer tasks are shown in Table 6.1 and are further elaborated in Figure 6.2. IWBBA naturally combines the strengths of white-box (human) and black-box (computer) that have been discussed earlier in Section 6.1.

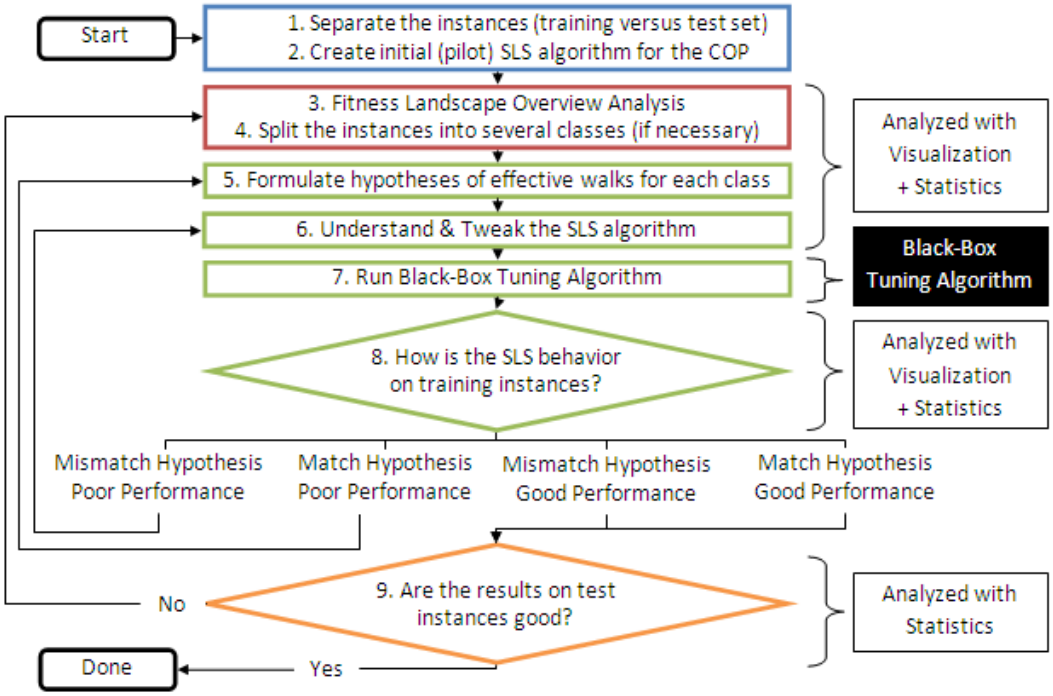


Figure 6.2: The Integrated White+Black Box Approach (see details below)

Steps 1-2: The algorithm designer separates training versus test instances from the available COP instances data. He also implements an SLS algorithm *that works* for the given COP (a pilot implementation on the training instances).

Steps 3-5: The algorithm designer executes some pilot runs to understand the COP fitness landscape(s) by answering questions posed in Section 4.2. He splits the instances into classes if significant differences are found. He then formulates *hypotheses* of effective walks for each class of instances.

Step 6: The algorithm designer creates experiments to answer questions posed in Section 4.3. He uses FLST visualization to observe how various SLS algorithms *actually* behave on some COP training instances. FLST visualization

fits our needs as this kind of information is hard to obtain without proper visualization. The obtained insights augment the algorithm designer's abilities to design better SLS algorithm or to invent new strategies/ideas.

Step 7: The white-box step in step 6 may also narrow down the configuration set. The algorithm designer can then pass the focused configuration set to a black-box tuning algorithm for even more performance.

Step 8: After step 6 and 7, analyze the tweaked and tuned SLS behavior on training instances. Does it match our hypothesis of effective walks and whether the performance is good? There are several possibilities:

If hypothesis matches and performance is good: The SLS algorithm is ok for the training instances. Do verification on test instances!

If does not match hypothesis but performance is good: Perhaps this is a new discovery? Verify if it is reproducible?

If hypothesis matches but performance is bad: Perhaps the hypothesis is wrong or the SLS algorithm is implemented wrongly.

If does not match hypothesis and performance is bad: Redesign the SLS algorithm again.

Step 9: Verify the results on test instances. If the SLS algorithm implementation produces satisfactory performance on test instances, stop. Otherwise, go back to step 3 as perhaps this is a case of over-fitting to training instances.

From a high level perspective, this integrated approach is not new (compare with Sequential Parameter Optimization [14] and also Software Engineering 'waterfall model')[2]. What is novel is how FLST visualization and statistics are used in the white-box steps (3-6, 8-9) and tuning algorithm in the black-box step (7).

6.3 Viz as a Black-Box Tuning Tool

In Chapter 5, we have presented Viz as a white-box visualization tool. To be more suitable for supporting IWBBA, especially in step 7, Viz needs to have an integrated black-box tuning algorithm. In Section 3.5.1, we have listed several more established tuning algorithms, e.g. GGA [9], ParamILS [70], F-Race [17], or CALIBRA [3]. However, integrating them into Viz system is not a straightforward task due to interfacing issues. Since the focus of this thesis is not on the design of automated black box tuning per se, we only provide simple support for a black-box tool.

To ease the fine tuning process, we build the following user interfaces (UIs). The first UI is the Viz **E**xperiment **W**izard (EW). The basic feature of Viz EW is a user-friendly interface (see Figure 6.3) for the user to prepare problem design (label A: add/remove COP instances, set/edit run time limit/BK OV/instance group for each instance), prepare an algorithm design (label B: add/remove SLS + configuration, edit configuration set, filter some configurations), customize experiment settings (label C: set number of replications, target OV), and observe the fine-tuning results (label D).

[2]Note that IWBBA is an iterative process. The results after performing steps 3-9 may produce a better SLS implementation, which in turns provide a more accurate FLST analysis in the next iteration.

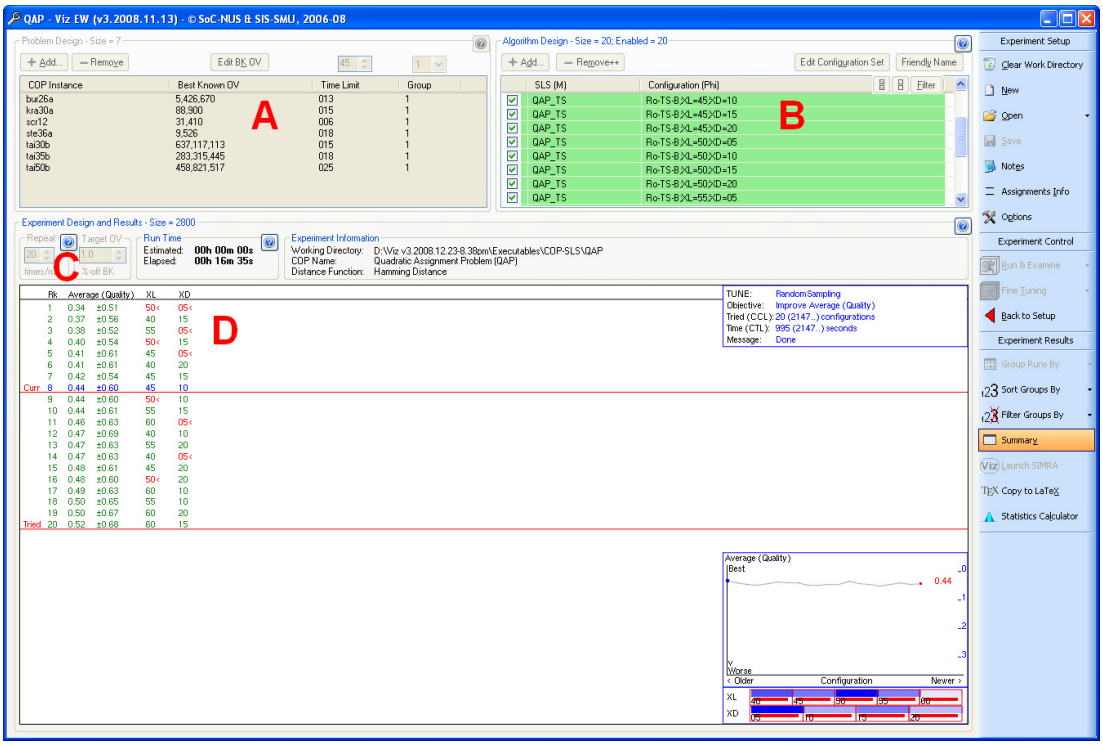


Figure 6.3: Viz v3.2008.11.13: The Experiment Wizard (EW)

The second UI is the configuration set editor. Here, the user can specify the configuration set by declaring the domain values of each SLS configurable part, see Figure 6.4. Then, Viz EW will generate the *full factorial design* of all possible configurations [102]. Notice that the size of the configuration set (denoted as $|CS|$) grows fast when domain values for the SLS configurable parts are added.

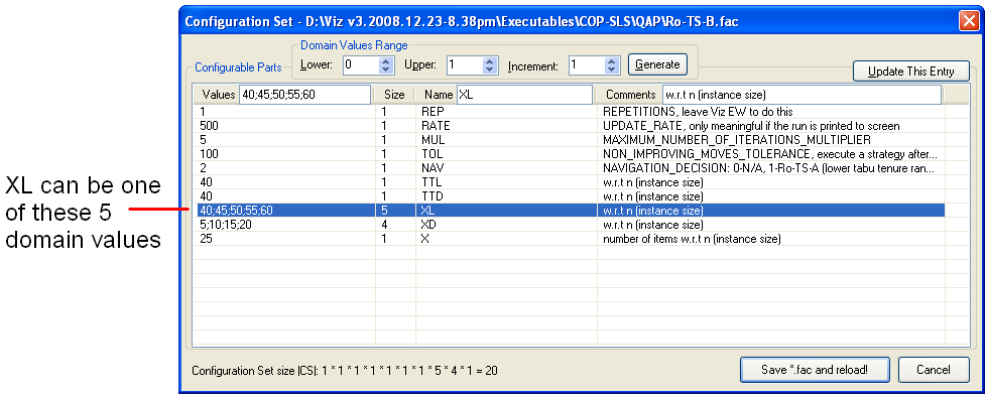


Figure 6.4: Configuration Set Editor

Since SLS development time is limited, Viz EW can only try a subset X (user adjustable) out of $|CS|$ possible configurations. For this, we use a simple *random sampling* algorithm that randomly picks X out of $|CS|$ possible configurations to be tried and returns the one that performs best on training instances[3]. When $|CS|$ is small enough, one can set $X = |CS|$ to set the random sampling algorithm to try all configurations in the full factorial design. This is the case in our experiments in Chapter 7.

[3]Tuning objectives like max/minimize average/total OV-qualities/runtimes, are user adjustable.

6.4 IWBBA Using Viz to Address User's SLS DTP

Figure 6.5 gives a general outline of how the IWBBA using Viz is typically performed on the SLS DTP.

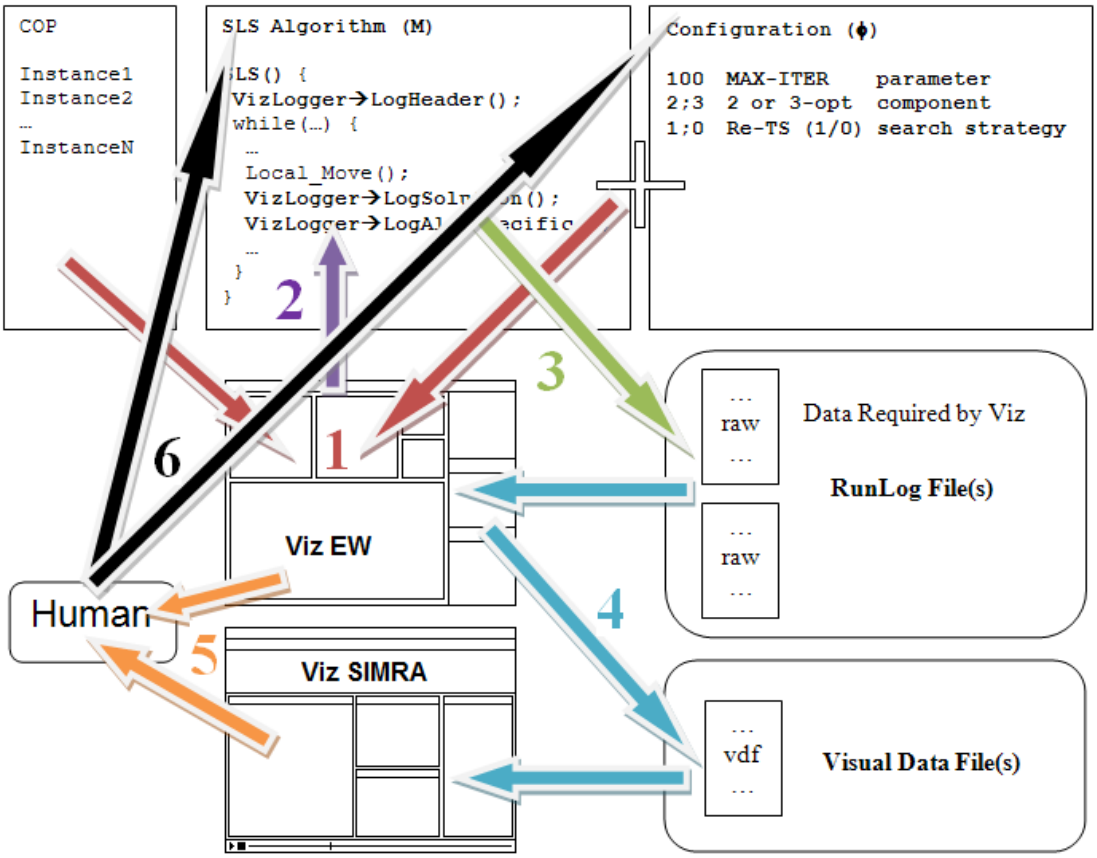


Figure 6.5: Overview of Viz Work Flow and Usage

Initially **(two red arrows with label 1)**, the algorithm designer selects COP training instances (problem design) and reserves the rest as test instances. He must supply the best known OV for each instance so that Viz can compute the SLS performance. He also selects the implementation of his SLS algorithm and select its configurations (algorithm design) using the configuration set editor shown in Figure 6.4.

Then, he invokes Viz EW **(purple arrow with label 2)** to execute the selected experiment design. If there are more than one configuration to try or there are several runs of the same SLS, Viz EW will try them based on the black-box *random sampling* strategy mentioned in Section 6.3.

When the SLS is running on the selected COP training instances, it logs the search information into 'RunLog' files **(green arrow with label 3)**. The algorithm designer must embed a simple logging mechanism into his SLS code that will record information like current solution structure, objective value, etc per SLS iteration. The log file format is described in the Viz's website (http://sls.visualization.googlepages.com).

Viz EW uses these RunLog files to generate the data for the FLST and other visualizations. For this, the algorithm designer must select a suitable distance function,

e.g. if his COP solution has bit-string structure or assignment, he may want to select *Hamming* distance; if his COP solution is a tour, bond distance may be appropriate; if his COP solution is a sequence, deviation distance may be considered, etc (see [123, 124]). Viz EW uses the selected distance function to obtain distance information. The other generic information required to build FLST visualization such as objective value and time are already recorded inside the log files. The next step is to save the processed data as Visual Data Files (VDFs) in order to avoid repeating this expensive task. These VDFs can be displayed in Viz SIMRA according to the playback speed selected by user **(three blue arrows with label 4)**.

The fine tuning results are presented directly in Viz EW window and the detailed VDFs can be played back in Viz SIMRA (as shown in Chapter 4 and 5). This is where the algorithm designer exercises his human strengths in understanding the visualization data. The information gained may inspire insights or further investigations which are essential for dealing with the SLS DTP **(two orange arrows with label 5)**.

The algorithm designer then uses the knowledge gained to improve the SLS algorithm design or to narrow down the configuration space **(two black arrows with label 6)**. Afterwards, he repeats the whole engineering cycle again, perhaps on different COP training instances, until satisfactory results are obtained.

6.5 Comparison with Existing Approaches

In Chapter 3, we have reviewed various white and black-box approaches for addressing the SLS Design and Tuning Problem. In Table 6.2, we present a subjective comparison of the differences of these existing approaches w.r.t IWBBA.

Approach	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**
Type of Method	B	B	B	B	B	B	B	W	W	W	W+B
Addressing type-1	H	M	M	E	E	E	E	H	H	H	E
Addressing type-2	H	M	M	M	E	E	E	H	H	H	E
Addressing type-3	¬	¬	¬	¬	¬	¬	¬	H	M	E	E
Ease of Usage	H	M	H	E	M	M	M	H	M	M	E

Table 6.2: Comparison of the Reviewed Approaches w.r.t IWBBA

Naïve Approaches: **A**: Ad-hoc Tuning, **B**: Brute Force Tuning,
Black-box Approaches: **C**: Meta SLS, **D**: CALIBRA [3], **E**: F-Race [18, 17, 12], **F**: ParamILS [69, 70], **G**: GGA [9],
White-box Approaches: **H**: Statistical Analysis, **I**: Human-Guided Search [7], **J**: Visualization of Search Landscape/Behavior [111, 136, 76, 86],
Integrated Approach: **K**: Integrated White+Black Box Approach.

Legends:
B: Black-box, W: White-box, W+B: Integrated Approach,
E: Easy, M: Medium, H: Hard, ¬: Not applicable.

6.6 Summary

1. Understanding SLS algorithm behavior on different COP fitness landscape helps the algorithm designer to design appropriate search strategies for those instances (better algorithm design). This recognizes the role of the human designer.

2. While the human is good in designing SLS algorithms, picking the best configuration for the SLS (i.e. fine-tuning) is better left to automated tuning.

3. Integrated White+Black Box Approach (IWBBA) is a natural combination of the strengths of both white+black box approaches – a man-machine approach. The detailed steps of IWBBA are presented in this chapter.

4. To facilitate IWBBA, we have designed Viz EW as a simple black-box tuning tool to complement the Viz SIMRA tool discussed in Chapter 5.

5. Adopting Viz to solve user's SLS DTP is relatively easy. What one's need to do is to follow the work flow for using the two Viz programs, i.e. add logging mechanism into the SLS code, select appropriate distance function, etc.

6.7 Looking Ahead

Now, we have the white-box (FLST and other SLS visualizations in Chapter 4 and 5) and black-box tools in Viz to apply IWBBA. What is left are the experimental evaluations. In Chapter 7, we present several successful experiments demonstrating IWBBA using Viz in addressing various SLS DTP scenarios.

Chapter 7

Experimental Results

It does not matter how beautiful your theory is.
If it does not agree with experiment, it is wrong!
— **Richard Feynman**

In this chapter, we present the results of applying IWBBA using Viz *to address three SLS DTP scenarios. These results have been reported in [59, 60, 61] (TSP), [57, 61] (QAP), and [58] (LABS Problem). However, all experiments have been re-done using the latest version of* Viz *(v3.2008.11.13) for consistent screen shots and results.*

7.1 Preliminaries

In Chapter 6, we have presented the Integrated White+Black Box Approach (IWWBA) for addressing the SLS Design and Tuning Problem (DTP) posed in Chapter 3. IWBBA consists of the white-box Fitness Landscape Search Trajectory (FLST) visualization (described in Chapter 4) and the black-box tuning algorithm. To facilitate IWBBA, we have build an SLS engineering suite called Viz. The white-box visualizations of Viz are described in Section 4.5 and Chapter 5. The black-box tuning tools of Viz are described in Section 6.3.

In this chapter, we present the results of applying IWBBA using Viz to three SLS DTP scenarios on three different Combinatorial Optimization Problems (COPs). Ideally, we should try IWBBA on non classical COPs that pose challenges to the existing SLS algorithms. The objective is to show that with IWBBA we can get more insights and develop better SLS algorithms for these non classical COPs. Indeed, we have one such case with our third scenario on the Low Autocorrelation Binary Sequence Problem (LABSP) [58]. However, since a non classical COP is harder to benchmark, we first show our results on two well known classical COPs: the Traveling Salesman Problem (TSP) and the Quadratic Assignment Problem (QAP). More details such as the literature review of other algorithms to solve these COPS are listed in Appendix

A.

To make this chapter concise, we have taken the liberty to present these experimental results mostly from the *final step viewpoint*. The FLST visualization screen shots shown in this chapter make use of the $APset$ from *both* good and bad runs in the entire development process. Except for several cases that are explicitly mentioned, we generally do not show the details of the actual development process, in which 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 the FLST visualization, see Section 3.7 of [57].

The time complexity to build the FLST visualization is dominated by the $O(|APset| * n)$ distance computations in `AP-Selection` and `Search-Position-Layout` Heuristics discussed in Chapter 4. To keep the visualization analysis time reasonable, we only run training instances up to $n = 50$ and set $|APset|$ to be max(25, min(50, $0.5 * n$)) which keeps $|APset|$ within a reasonable range of [25..50] for any instance size n.

The computer used for the TSP and QAP experiments in this chapter is a 2.83 GHz Core2 Quad PC with 3.25 GB RAM. However, for the LABSP experiments, we use *multiple* PCs with different specifications (details in Section 7.4). This is to facilitate runtimes comparisons with the results of other algorithms in the literature.

We have to remark that the experiment results shown in this chapter are obtained under no specific development time constraints.

Videos and more pictures of the experiments from this chapter are available in VIZ website: `http://www.comp.nus.edu.sg/~stevenha/viz/results.html`. It may be clearer to view the animation than the static pictures printed in this thesis.

The general outline of the next three sections is shown below. The outline follows the IWBBA steps in Chapter 6 (Figure 6.2), except that the process is not iterative as the experiment results are shown from the final step viewpoint.

1. Stating the Experiment Objectives
2. Describing Formal Problem Description of the COP
3. Elaborating the Experimental Setup
   - Selecting benchmark instances
   - Setting FLST quality measures, and
   - Describing the baseline algorithm taken from literature
4. Performing Fitness Landscape Search Trajectory Analysis
   - Analyze fitness landscape of the COP, split instances into classes if necessary
   - Analyze search trajectory of the baseline algorithm for each class
   - Stating hypothesis of better walks
5. Redesign and Fine Tune the SLS Algorithm
   - White-box: use the insights to improve the SLS algorithm design
   - Black-box: perform fine tuning on the focused configuration set
6. Verify Results on Test Instances

7.2 Traveling Salesman Problem

Our earlier versions of these experiments have been published in [59, 60, 61].

7.2.1 Experiment Objectives

This section also serves as a *tutorial* on applying IWBBA using Viz. The results by themselves are not new – they have been found and analyzed by other researchers [133] using the existing white-box techniques (FDC analysis to detect the 'Big Valley' and RTD analysis to detect search stagnation). However, they have never been visualized in this way before. We show that Viz can also give researchers the same insights in a more intuitive fashion.

7.2.2 Formal Problem Description

The input for the Traveling Salesman Problem (TSP) is a complete, weighted graph $G(V, E)$ with V being the set of vertices, representing the cities, and E being the set of edges fully connecting the vertices. Each edge is assigned a value d_{ij}, the length of $edge(i, j)$, that is, the distance between cities i and j, with $i, j \in V$. The objective of TSP is to find a *minimal length Hamiltonian circuit* of the graph where a Hamiltonian circuit is a closed tour $s = \{s_0, s_1, \ldots, s_{n-1}\}$ visiting each of the vertices of G exactly once and s minimizes the objective function:

$$g(s) = \sum_{i=0}^{n-2} d_{s_i s_{i+1}} + d_{s_{n-1} s_0} \tag{7.1}$$

TSP is an NP-Complete problem [47].

7.2.3 Experimental Setup

Benchmark Instances

For the TSP experiments in this thesis, we use the training and test instances shown in Table 7.1. All instances are taken from TSPLIB [119, 143]. They are symmetric Euclidean 2D instances where the distances between the cities are Euclidian distances and independent of the edges direction, i.e. $d_{ij} = d_{ji}$ for every pair of vertices. The selected instances have sizes between 51 and 280.

Training Instances (4 Instances)		Test Instances (21 Instances)						
eil51	pr76	berlin52	eil76	st70	rat99	kroB100	kroC100	rd100
kroA100	lin105	eil101	pr107	pr124	bier127	ch130	ch150	kroA150
		kroB150	u159	rat195	d198	tsp225	gil262	a280

Table 7.1: Training and Test Instances for the TSP experiments

FLST Quality Measures

As the quality of good TSP solutions is known to be near the Best Known Objective Value (BK OV), we define these quality measures: "Label/(icon in the FLST visual-

ization): $[a-b)$", where a & b are percentage-off from BK OV $\Rightarrow$ **Good/O**: $[0-1)$, **Medium/△**: $[1-2)$, **Bad/□**: $[2-3)$, and **VeryBad/.**: $[\geq 3)$.

To measure the distance between TSP tours/solutions, we use the bond distance a.k.a permutation/edge distance defined below. It measures the distance of two permutation based solutions where the order of items does not matter but the edges/links between adjacent items do. In TSP, we want to know the number of different edges between the two solutions, e.g. $d(\{\text{A-B-C-D}\}, \{\text{B-C-D-A}\}) = 0$ and $d(\{\text{A-B-C-D}\}, \{\text{B-D-C-A}\}) = 2$. This bond distance satisfies the triangle inequality property that is useful for the FLST visualization [123]. Bond distance is computed as follows:

`bond-distance` $d(s_1, s_2) = n - \sum_{k=0}^{n-1}$ `common-edge`$_k$

`common-edge_k = 1, if e`$(s_{1_k}, s_{1_{(k+1)\%n}})$ `or e`$(s_{1_{(k+n-1)\%n}}, s_{1_k})$ `exists in` s_2
`= 0, otherwise`

As this distance function is frequently used in the FLST visualization, its computation must be efficient! Bond distance can be computed in $O(n)$ with pre-processing. In one pass ($O(n)$ time), record n edges of the 1st tour in a Hash Table HT. Then, in another pass, count the number of edges of the 2nd tour that are in HT (also in $O(n)$ time).

Baseline Algorithm

The initial baseline SLS algorithm for this experiment is the Iterated Local Search (ILS) by [133]. The pseudo-code and the configurable parts of ILS are shown in Figure 7.1. A search strategy given in line 14-15 are discussed in Section 7.2.5.

```
ILS(n)
 1  CurS ← BF ← LS(InitS(n))
 2  i ← c ← 0 // i = iteration counter, c = non improving moves counter
 3  while i < MAXITR
 4  do NewS ← LS(Ptb(CurS)) // Ptb is one 4-Opt double bridge move
 5     if AccC = BETTER
 6        then if g(NewS) < g(CurS)
 7                then CurS ← NewS // Only move if NewS is better (BTR)
 8        else  CurS ← NewS // Always move to NewS (RW)
 9     if g(CurS) < g(BF)
10        then c ← 0
11             BF ← CurS
12        else  c ← c + 1
13     i ← i + 1
14     if c > TOL% * n // line 14-15 are discussed in Section 7.2.5
15        then CurS ← LS(FDD-DIVERSIFICATION(CurS))
16  return BF
```

Configurable Part	Initial Choice	Remark
MAXITR (MaxIteration)	n^2	Considered as a short run
AccC (AcceptanceCriteria)	Better	Only move to a new local optimum if it is better
InitS (InitialSolution)	N-N	$O(n)$ Nearest Neighbor Heuristic
LS (LocalSearch)	2-Opt	$O(cn)$ Swap 2-Edges with help of candidate list
g (ObjectiveFunction)	delta	$O(1)$ Old OV - 2 deleted edges + 2 added edges
Ptb (Perturbation)	4-Opt	Double Bridge Move [133]

Figure 7.1: Code and initial configuration of **ILS** for TSP

ILS for TSP makes use of a 4-Opt double bridge perturbation move[1] (Ptb in line 4), then a local search that uses 2-Opt swap edge heuristic[2] transforms the perturbed solution to its 2-Opt local optimum (LS in line 4). If the acceptance criteria (**AccC** in line 5) is set to 'Better', then ILS will only move to the new local optimum if it has a smaller OV than the current local optimum $CurS$ (line 6-7), otherwise the ILS stays at $CurS$. This procedure is iterated until the number of iterations has reached **MAXITR**.

7.2.4 Fitness Landscape Search Trajectory Analysis

Fitness Landscape Analysis: TSP has the 'Big Valley' Property

There are up to $n!/2n$ distinct TSP tours/solutions for a TSP instance of size n. Although the search space is very big, previous research (e.g. [45, 133, 89, 68]) has shown that TSP instances have the 'Big Valley' property: good solutions (i.e. TSP local optima including the global optima) lie in only a small region of the search space.

Figure 7.2 illustrates why TSP has the 'Big Valley' property: high quality TSP tours tend to be similar! Bond distances between these high quality TSP tours are small, bunching them in a small region in the fitness landscape.

This 'Big Valley' property can be intuitively shown using the FLST visualizations in VIZ (FLO mode[3]). FLST visualizations of the fitness landscape of three TSP instances (pr76, kroA100, and lin105) are shown in Figure 7.3. The visualization error (`errAP`) in Figure 7.3 is small: 0.07, 0.04, 0.03 for pr76, kroA100, and lin105, respectively.

In Figure 7.3, we see that the better[4] local optima are clustered around the middle of the screen. Since the Best Found BF solution is drawn in the middle of the screen in the FLST visualization and one grey ring is 10 distance units, we observe that the *majority* of the better local optima are not too far from the BF solution (average/maximum distance = 20/76 for pr76, 25/100 for kroA100, and 25/105 for lin105 which is $\approx 30\% * n$). The pairwise distances between local optima are also small. This results in a low $DiversityIndex$: 0.27, 0.19, 0.18, for pr76, kroA100, and lin105, respectively, which helps the spring layout algorithm `NEATO` to layout these APs with only small `errAP`. When zoomed in, we observe that the position of **Good/O** or **Medium/△** APs are closer to the BF AP. Further out, the solution qualities drop (**Bad/□** and **VeryBad/.**).

This 'Big Valley' property can also be checked using the Fitness Distance Correlation (FDC) analysis which shows whether TSP solutions that have small TSP tour lengths (better OV/fitness) are also closer to the BF solution. In Figure 7.4, we observe that all the FDC scatter plots have positive correlation: positive linear regression lines and high r_{FDC} (≥ 0.65 in our training instances). This means that the OV can give good guidance: when the SLS algorithm moves to a better solution, it is usually closer to the BF solution. Notice that most good local optima are close to the BF solution.

[1] The 4-Opt double bridge perturbation move cuts the current tour at four random positions: $A - B - C - D$. These four sub-tours are then reconnected in the order of $A - D - C - B$.

[2] The 2-Opt swap edge move cuts the current tour at two positions: $A - B$. These two sub-tours are then reconnected in the order of $A - B'$ where B' is B in the reverse direction.

[3] Recall Fitness Landscape Overview (FLO) mode in Section 4.5.3.

[4] Recall that the worse ones have been filtered away by the AP selection heuristic in Section 4.5.2.

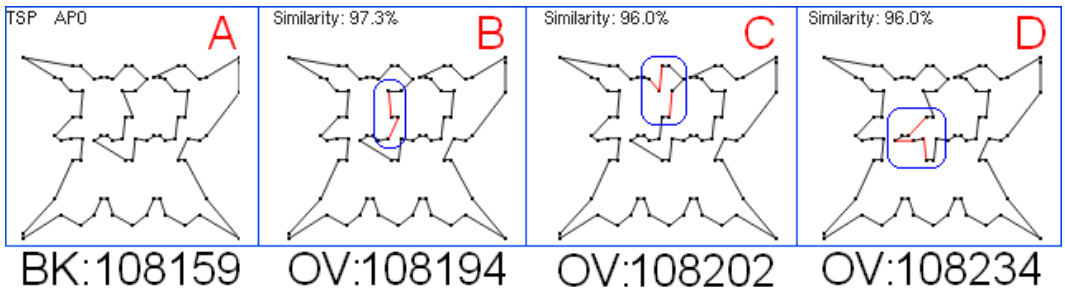


Figure 7.2: A): Optimal Tour of 'pr76'; B-D): 3 Other 'Similar' Local Optima. These screen shots are taken directly from Viz Problem-Specific visualization for TSP.

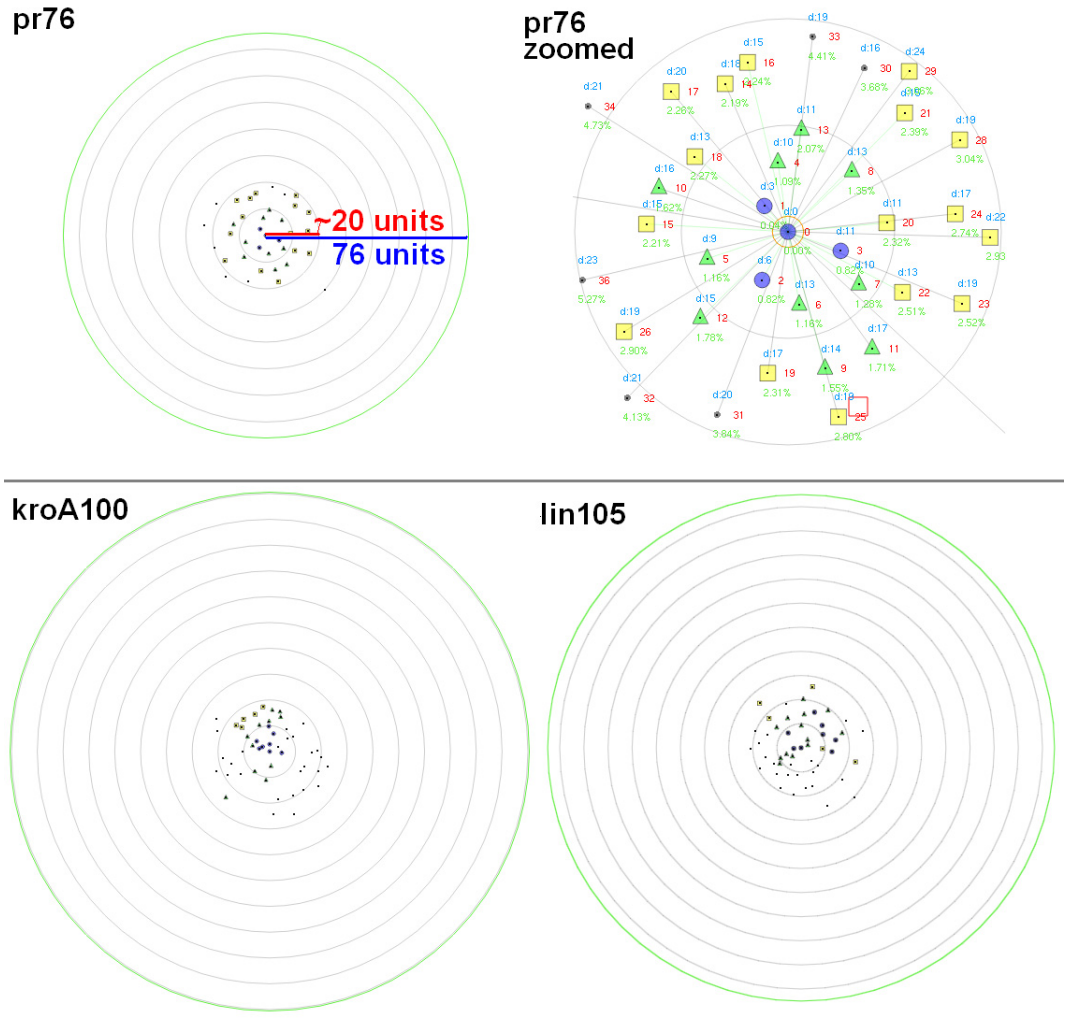


Figure 7.3: The 'Big Valley' in TSP instances are observable using FLST Visualization. These screen shots are taken directly from Viz FLST visualization.

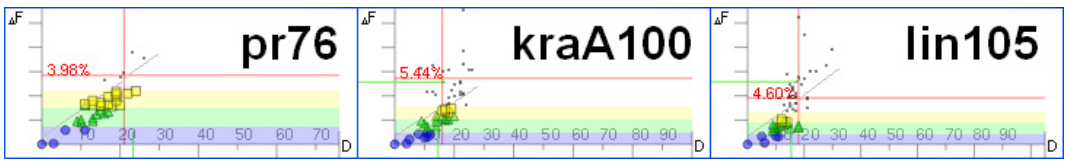


Figure 7.4: The 'Big Valley' in TSP instances are observable using FDC Analysis. These screen shots are taken directly from Viz FDC visualization. $r_{FDC}(\text{pr76}) = 0.82$, $r_{FDC}(\text{kroA100}) = 0.65$, $r_{FDC}(\text{lin105}) = 0.71$.

Search Trajectory Analysis and Hypothesis of Better Walks

As TSP instances have the 'Big Valley' property, a good walk is an intensification around the 'Big Valley' region. Although an SLS algorithm does not know if it is currently searching in the 'Big Valley' region, there is a heuristic that if the SLS algorithm finds a better local optimum, it is moving closer to the 'Big Valley' region (and the global optima).

We analyze the ILS search trajectories in Section 7.2.5 below.

7.2.5 ILS for TSP

White-Box Step: Avoiding Stagnation in ILS for TSP

The ILS algorithm shown in Figure 7.1 is an example of an SLS algorithm that has been designed to exploit the 'Big Valley' property of TSP. The key is in lines 5-7. Here, the ILS is prevented from moving to a new TSP local optimum $NewS$ produced by a perturbation and local search in line 4 if $NewS$ is found to be not better than the current local optimum $CurS$. This prevents the ILS from moving to a worse local optimum – which is considered as moving away from the global optima according to the 'Big Valley' property. Only if $NewS$ is *better* than $CurS$, will ILS accept $NewS$ – which is expected to be closer to the global optima according to the 'Big Valley' property. We call this ILS as ILS_{BTR}.

However, if the parameter **AccC** is set to RandomWalk, then the ILS will execute line 8, i.e. ILS always accepts $NewS$ at every iteration. We call this ILS as ILS_{RW}.

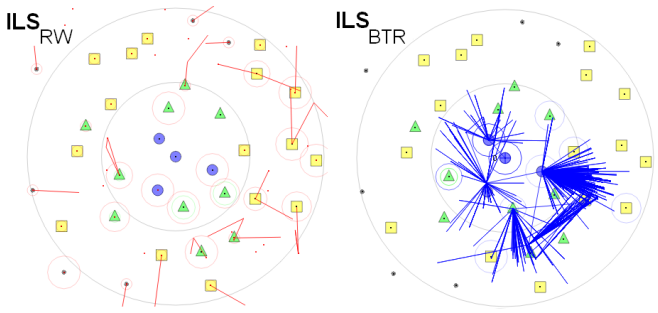


Figure 7.5: Search Trajectory Coverage + Detail of **ILS_{RW}** (left) versus **ILS_{BTR}** (right) on the same Fitness Landscape of a TSP Instance: 'lin105' after ≈ 1800 iterations. Note that the $APset$ used in this FLST visualization is shown from the final step viewpoint. The APs are mostly from the better ILS_T (discussed below) and ILS_{BTR} runs.

FLST visualization (SCO + STD mode[5]) shown in Figure 7.5 intuitively explains why ILS_{BTR} is a better strategy than ILS_{RW}. We can identify that most of the time ILS_{RW} (**red lines and circles** on the left) is straying away from the middle of the screen where the 'Big Valley' region is located. ILS_{RW} trajectory is only occasionally shown to be near the center of the screen. We can say that ILS_{RW} does too much diversification and thus does not search in the correct place. In contrast, ILS_{BTR} (**blue lines and circles** on the right) stays focused in the 'Big Valley' region at the middle of the screen. The

[5]Recall Search Coverage Overview (SCO) & Search Trajectory Detail (STD) mode in Section 4.5.4.

behavior of ILS_{BTR} that frequently rejects $NewS$ if it is worse than $CurS$ is clearly seen in the visualization in form of 'star patterns': the center of the star is $CurS$ and the spiky ends[6] of the star are the rejected $NewS$ solutions. These star patterns are shown to be appearing in the correct place: near the middle of the screen.

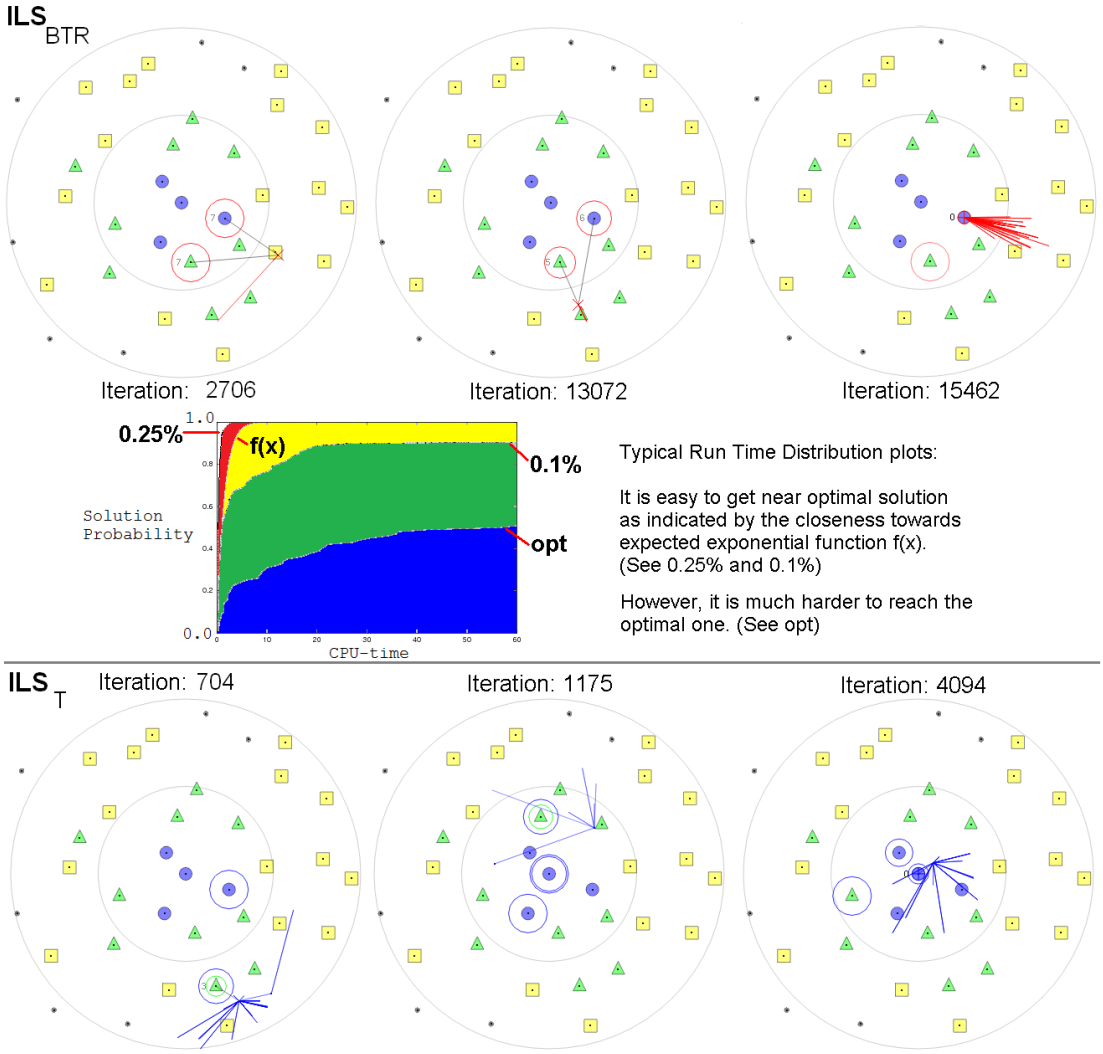


Figure 7.6: Visualization of TSP Fitness Landscape and ILS Behavior on lin105. We also provide the typical Run Time Distribution plot found in [133].

However, although we have observed in Figure 7.5 that ILS_{BTR} is already making sufficient use of the 'Big Valley' property, this is not enough. A longer[7] playback on FLST visualization shown in Figure 7.6 (first row) reveals that ILS_{BTR} (now colored with **red lines and circles**) is actually stuck in similar fitness landscape region on a large number of iterations (2706 to 13072). This exceeds the max $105^2 = 11025$ iterations in this experiment which implies that ILS_{BTR} often reports sub-optimal results in short runs. This observation of 'ILS_{BTR} being stuck' is more clearly seen using

[6]Notice that most spiky ends do not point to another AP in the visualization but to the blank spaces around few APs. This is because the $NewS$ solution is mostly not inside the $APset$. Thus its position must be approximated using the STD mode discussed in Section 4.5.4.

[7]Recall that Figure 7.5 only shows up to 1800 iterations.

an animation where the 'star patterns' are staying in the same place for a substantial number of iterations. This phenomenon is observable on the other training instances too.

As mentioned earlier, Stützle and Hoos [133] have arrived at this conclusion using the Run Time Distribution (RTD) analysis (see Section 3.5.2). The RTD plot in the middle of Figure 7.6 is taken from [133] and shows the *typical* solution probability of ILS_{BTR} with 2-Opt moves on a TSPLIB instance. The RTD plot shows that reaching good TSP solutions is easy as solution probability is near 1.0 if we only aim for a sub-optimal solution. But, reaching optimal ones is hard as there is an observable stagnation behavior in its RTD plot: solution probability to reach optimal solution is not close to 1.0 even if we use *much more* CPU time. However, the RTD plot cannot tell much beyond this information (see Figure 4.13 and its corresponding text).

In [133], the authors suggested 'a fix' by using a stronger diversification than the double bridge move, FDD-DIVERSIFICATION, that does a *controlled diversification* around the 'Big Valley' region if a number of iterations $\textbf{TOL}\% * n$ has elapsed without any improvement. Basically, FDD-DIVERSIFICATION is a strategy to generate sufficiently far local optimum from $CurS$ but yet still good enough to be in the 'Big Valley' region (although not necessarily better than current local optimum $CurS$). It works by generating some local optima (perturbing $CurS$ + local search), pick the fittest ones, and among the fittest, the furthest one from $CurS$. In this section, we will intuitively show using FLST visualization how ILS_{BTR} is improved by using this search strategy.

Black-Box Step: Fine-Tuning

Now we have a tuneable parameter **TOL**. White-box approaches are not suitable to tune **TOL** as the differences are too hard to be noticed. It is better to use a black-box tuning algorithm to tune **TOL**. We try these values **TOL** = {100, 150, 200, 250, 300} with all other parameters set as in Figure 7.1. We obtain **TOL** = 200 as the fittest setting on the training instances. We call the resulting SLS strategy as $\text{ILS}_{T(weaked)}$.

The improved behavior is observable in Figure 7.6, label ILS_T (second row). ILS_T (**blue lines and circles**) is now able to escape from several local optima attractors easier than ILS_{BTR}: the 'star patterns' in ILS_T do not stay in a fitness landscape region for too long as FDD-DIVERSIFICATION throws ILS_T sufficiently far from its previous position (yet still around the 'Big Valley' region) at every $\textbf{TOL}\% * n$ non-improving moves. We observe that with this search strategy, ILS_T progresses closer towards the center of the screen in typically less[8] number of iterations than ILS_{BTR}. This strategy works because there may be simpler (2-Opt) paths towards the better solutions (located in the middle of screen) from *other* local optima rather than from $CurS$. Thus, the ILS should rather give up and try other local optima if such an improving path cannot be easily found from $CurS$. Since both ILS variants are limited to n^2 iterations only,

[8] In the example in Figure 7.6, ILS_T reaches optimal solution at iteration 4094, compared with ILS_{BTR} that requires > 11025 iterations.

ILS_T has a better chance of finding better solutions than ILS_{BTR}.

7.2.6 Results on Test Instances

Table 7.2 shows the performance of ILS variants (ILS_{RW}, ILS_{BTR} and ILS_T) on test instances. Each algorithm is run on a particular instance for n^2 iterations. As there is a random element, this process is repeated **20 times** to obtain the average percentage-off $\bar{x}$ and standard deviation σ from BK OV. We observe that the **bold entries** (the best result on a particular row/TSP test instance) are mostly appearing in ILS_T column, which also has the best average values among the three columns.

Wilcoxon signed-ranks test detects that the results of the baseline algorithm ILS_{BTR} is *clearly* better than ILS_{RW} (21 pairs, $p < 0.05, T = 0.0, V = 67$) and the results of the tweaked algorithm ILS_T is *significantly* better than the baseline algorithm ILS_{BTR} (21 pairs, $p < 0.05, T = 28.0, V = 67$).

Test Instances

Instance	**Iters**	BK **OV**	$\textbf{ILS}_{RW}$		$\textbf{ILS}_{BTR}$		$\textbf{ILS}_T$	
			$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ
berlin52	2704	7542	2.44	0.95	0.49	0.98	**0.00**	0.00
eil76	5776	538	3.90	0.87	0.60	0.62	**0.50**	0.38
st70	4900	675	2.37	0.56	0.91	0.71	**0.22**	0.30
rat99	9801	1211	3.69	0.82	0.66	0.83	**0.21**	0.23
kroB100	10000	22141	2.01	0.59	0.42	0.48	**0.11**	0.16
kroC100	10000	20749	2.17	0.52	0.29	0.37	**0.10**	0.16
rd100	10000	7910	3.32	0.56	0.96	0.80	**0.15**	0.21
eil101	10201	629	5.48	0.60	1.39	0.74	**1.21**	0.74
pr107	11449	44303	8.74	4.62	**3.56**	2.43	4.45	2.08
pr124	15376	59030	0.88	0.45	0.71	0.79	**0.07**	0.08
bier127	16129	118282	2.46	0.40	0.29	0.19	**0.24**	0.12
ch130	16900	6110	3.02	0.59	0.87	0.41	**0.41**	0.36
ch150	22500	6528	3.71	0.57	0.60	0.32	**0.37**	0.16
kroA150	22500	26524	3.87	0.67	0.60	0.65	**0.46**	0.36
kroB150	22500	26130	3.62	0.45	0.64	0.56	**0.45**	0.33
u159	25281	42080	3.92	0.85	0.84	0.68	**0.22**	0.27
rat195	38025	2323	4.39	1.16	0.88	0.47	**0.81**	0.39
d198	39204	15780	11.49	2.90	**1.36**	0.68	1.37	0.52
tsp225	50625	3916	5.55	0.66	0.70	0.60	**0.66**	0.42
gil262	68644	2378	5.70	0.57	**0.87**	0.42	1.00	0.38
a280	78400	2579	8.87	1.07	1.39	0.85	**1.18**	0.44
		Average ⇒	4.36		0.91		**0.68**	

Table 7.2: ILS Variants Results

In summary, we shown in this section the 'Big Valley' fitness landscape structure of TSP instances and why the standard Iterated Local Search ILS_{BTR} (by [133]) experiences stagnation in its behavior can be explained in a more intuitive manner using FLST visualization than by using existing approaches (e.g. the FDC and RTD analysis). We also illustrate and intuitively compare the improved ILS_T (also by [133]) has better search trajectory than ILS_{BTR}. This shows an additional use of FLST visualization – to explain/teach SLS algorithms.

7.3 Quadratic Assignment Problem

The white-box part of QAP experiments has been published in [57] and in [61] (with the black-box part). Note that the results in this section are improved over [57, 61] as we use a more diverse benchmark instances beyond the Taillard instances.

7.3.1 Experiment Objectives

In this scenario, our objective is to get a good performing SLS algorithm for the QAP *with a limited number of iterations*. This is because there exist good SLS algorithms for QAP that can find BK OV of many QAP instances in QAPLIB [22, 114] with longer runs. To prevent the *ceiling effect*, where all runs reach the BK OV, we have fixed the number of iterations of each SLS run to be quite 'small': $5 * n^2$ iterations, where n is the instance size. Thus, not all runs are expected to reach the BK OV (especially for larger ones) and the better algorithms on such short runs can be distinguished.

We also show that the FLST visualization can help the algorithm designer to spot different classes of COP instances and design proper search strategies for each class. The improved SLS designs are then fine-tuned for more performance.

7.3.2 Formal Problem Description

The input for the Quadratic Assignment Problem (QAP) is two $n \times n$ matrices $A = (a_{ij})$ and $B = (b_{ij})$. Typically, matrix A contains flow information between facilities and matrix B contains distance information between facilities. The objective of QAP is to find a permutation s of $\{0, 1, 2, \ldots, n-1\}$ over all possible permutations in the search space which minimizes the objective function:

$$g(s) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} a_{s_i s_j} b_{ij} \tag{7.2}$$

This formulation is called Koopmans-Beckmann QAP [140] and is $\mathcal{NP}$-Complete [47].

7.3.3 Experimental Setup

Benchmark Instances

Training Instances (14 instances)		Test Instances (20 instances)			
nug20	bur26a	nug15	tai40a	bur26b	tai25b
rou15	kra30a	nug25	tai60a	bur26c	tai40b
sko42	scr12	nug30	wil100	kra30b	tai60b
tai30a	ste36a	rou20		kra32	
tai35a	tai30b	sko49		scr20	
tai50a	tai35b	sko56		ste36b	
wil50	tai50b	tai25a		ste36c	

Table 7.3: Training and Test Instances for QAP experiments

These benchmark instances are taken from QAPLIB [114]. There are 135 QAP instances from 15 authors in QAPLIB. We take 34 instances from 9 authors and split them into training and test instances as in Table 7.3.

FLST Quality Measures

We define QAP solution quality measures in a similar way to TSP ⇒ **Good/O**: $[0-1)$, **Medium/△**: $[1-2)$, **Bad/□**: $[2-3)$, and **VeryBad/.**: $[\geq 3)$.

A QAP solution is an assignment. $s[i] = j$ means facility i is located at position j. There is no direct connection between $s[i]$ and its neighbors ($s[i-1]$ and $s[i+1]$) $\forall i \in \{1, n-2\}$. The $O(n)$ Hamming distance defined below is appropriate[9] for QAP as it measures the distance of two permutation/bit string based solutions where the position/order of the items matter. In QAP, we want to know the number of facilities that are located in different location between the two solutions, e.g. d($\{0,1,2,3\}$, $\{0,1,2,3\}$) = 0 and d($\{0,1,2,3\}$, $\{2,1,0,3\}$) = 2. Hamming distance is computed as follows:

$$\texttt{hamming-distance}\ d(s_1, s_2) = \sum_{k=0}^{n-1} \texttt{mismatch}_k$$
$$\texttt{mismatch}_k \texttt{ = 1, if } s_{1_k} \neq s_{2_k}$$
$$\texttt{0, otherwise}$$

The Hamming distance also satisfies the triangle inequality property that is useful for the FLST visualization [123].

Baseline Algorithm

The initial baseline SLS algorithm for this experiment is the Robust Tabu Search (Ro-TS) by [137] which has been shown to give good performance on QAP. Ro-TS is a TS that frequently changes its Tabu Tenure within a predetermined range (details in Appendix B). We implemented a *variant* of Ro-TS called **Ro-TS-I** based on [137]. The pseudo-code and configurable parts of **Ro-TS-I** are shown in Figure 7.7.

Ro-TS-I starts from an InitialSolution (a random assignment). Then, it generates feasible neighbors which is $CurS$ with facilities i and j swapped (the $O(n^2)$ 2-Opt Neighborhood in line 5) if this pair $i-j$ has not been exchanged in the last $TT\% * n$ iterations (TabuTable in line 5) or aspired (AspirationCriteria in line 5). Ro-TS-I picks the best solution among these neighbors – which may not always be better than the current $CurS$ (line 6), moves there, and sets the recently applied move to be tabu for the next $TT\% * n$ iterations (line 7). Ro-TS-I utilizes the $O(1)$ incremental OV computation mentioned in [137] to speed up the computation. Additional search strategies given in line 13-19 are discussed in Section 7.3.5 and 7.3.6.

[9]The choice of a particular distance function depends on the nature of the COP being attacked and the elements that constitute a meaningful notion of distance within the fitness landscape. Otherwise the results will be inaccurate, e.g. bond distance is appropriate for TSP but not for QAP as edges in QAP solutions do not mean anything. Hamming distance is more appropriate for QAP. However, if we use Hamming distance to measure distance between TSP solutions, the resulting distances will be too far, and can be misleading (tour {1-2-3-4} and {2-3-4-1} are the same tours but Hamming distance returns $d = 4$ (maximum distance) in this case).

```
Ro-TS-I(n)
 1  CurS ← BF ← InitialSolution(n)
 2  i ← c ← 0 // i = iteration counter, c = non improving moves counter
 3  TT ← random([TTL..(TTL + TTD)])
 4  while i < MAXITR
 5  do Neighbor(CurS) ← Neighborhood(CurS, n, TabuTable, AspirationCriteria)
 6     CurS ← best(Neighbor(CurS))
 7     Update TabuTable by TT% ∗ n
 8     if g(CurS) < g(BF)
 9       then c ← 0
10            BF ← CurS
11       else c ← c + 1
12     i ← i + 1
13     if c > TOL% ∗ n
14       then c ← 0
15            if strategy = Ro-TS-I or Ro-TS-A // line 15-16 are explained in ...
16              then TT ← random([TTL..(TTL + TTD)]) // ... Section 7.3.5
17            if strategy = Ro-TS-B // line 17-19 are explained in Section 7.3.6
18              then X ← random([XL..(XL + XD)])
19                   CurS ← RuinAndRecreate(CurS, X)
20  return BF
```

Configurable Part	Initial Choice	Remark
MAXITR (MaxIteration)	$5 * n^2$	Considered as a short run
TTL (TTLow)	90	The Tabu Tenure range in [137]
TTD (TTDelta)	20	Similar as above
TOL (Tolerance)	200	Waiting time before executing strategy as in [137]
XL (XLow), **XD** (XDelta)	N/A	Explained in Section 7.3.6
InitialSolution	Random	Produce a random assignment
Neighborhood	2-Opt	$O(n^2)$ (Swap) move operation for QAP
g (ObjectiveFunction)	delta	Compute delta of OV in $O(1)$ as shown in [137]
TabuTable	pair $i-j$	Item i cannot be swapped with item j for TT steps
AspirationCriteria	Better	Override tabu if move leads to a better solution
SearchStrategy	Ro-TS-I	Change TT within $[0.9 \ldots 1.1]n$ after $2n$ steps [137]

Figure 7.7: Code and initial configuration of **Ro-TS-I** for QAP

7.3.4 Fitness Landscape Search Trajectory Analysis

Fitness Landscape Analysis: ≥ 2 QAP Fitness Landscape Types

We apply the FLO mode of FLST visualization on the training instances. In Figure 7.8, we see that the APs in {sko42, tai30a, ste36a, tai30b} are mostly spread[10] throughout the fitness landscape. Many APs are almost touching the **green border** (the maximum distance). The *DiversityIndices* of these training instances are near 1.0 as good QAP local optima can be very different. For example, *DiversityIndex* is 0.94 in tai30a (0.91 in tai30b). This implies that most of the 25 APs in tai30a FLST visualization have distance 28 out of 30 w.r.t each other. It is difficult to obtain small layout error `errAP` with such distance constraints (see Section 4.5.3).

Although the location of the APs are spread out, the quality of the APs in {sko42, tai30a} are more 'uniform' (most are **Good/O** or **Medium/△** APs and no **VeryBad/.** APs) than {ste36b, tai30b} (all AP quality types exist with many **VeryBad/.** APs).

[10] We remark that this information is not easy to be seen in FDC scatter plot, see Figure 4.8.

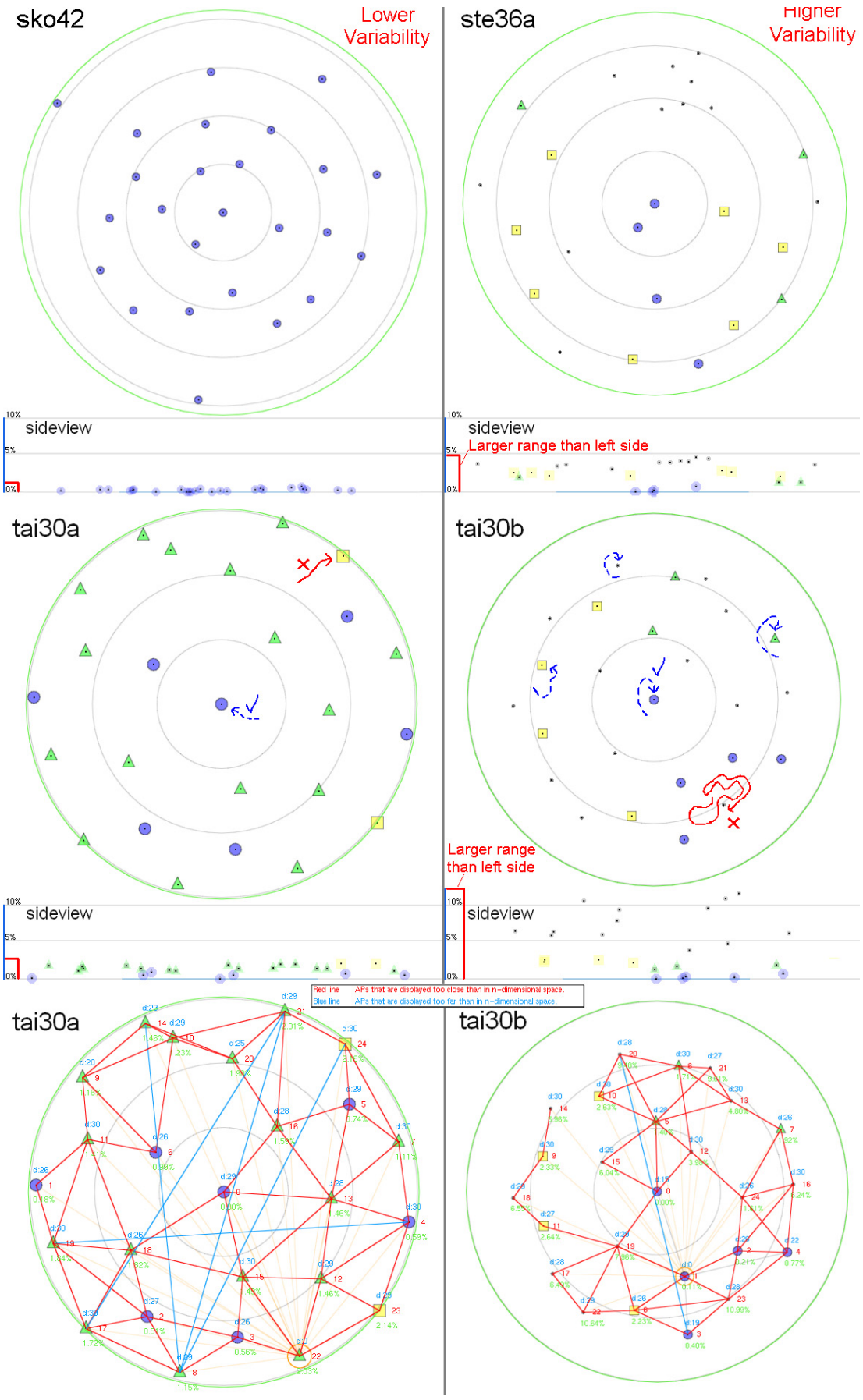


Figure 7.8: Fitness Landscape Overview of {sko42, tai30a} and {ste36a, tai30b}. Note the OV gap between the two types. Visualization error is high (elaborated in the text).

The 'sideview' mode shows this OV variability in a clearer manner: observe the **red range** that highlights the quality of APs w.r.t the BK OV. The instances on the right side of Figure 7.8 typically have a larger range than the ones on the left side.

A working hypothesis from this observation is that there are *at least* **two classes of QAP instances**. Others researchers (e.g. [89, 68]) are also aware of this and have used the dominance of a matrix $dm(matrix)$ (defined below) to classify QAP types.

$$avg(matrix) = \frac{\sum_{i=0}^{n-1}\sum_{j=0}^{n-1} matrix_{ij}}{n^2} \tag{7.3}$$

$$dm(matrix) = 100.0 * \frac{var(matrix)}{avg(matrix)} = \frac{\frac{\sum_{i=0}^{n-1}\sum_{j=0}^{n-1}(matrix_{ij}-avg(matrix))^2}{n^2-1}}{avg(matrix)} \tag{7.4}$$

Further observations link the smoothness/low variability (labeled as type A) and ruggedness/high variability (type B) of AP quality seen in the FLST visualization with the uniformity/low dominance (type A) and non-uniformity/high dominance (type B) property of their data matrices. Thus, classification of training instances can be done by observing the FLST visualization or the dominance of any of the two input matrices.

For classifying test instances, we *assume* if one *training* instance from an author is classified as a certain type[11], then classify other *test* instances from the same author to the same type. For example: taixxa instances are described in QAPLIB as *uniformly generated*; taixxa training instances are classified as type A; thus all other taixxa test instances that we do not use as training instances should belong to type A too.

The instances used in our experiments are now further classified as in Table 7.4.

Type A Instances				Type B Instances			
Training	dm(A)	dm(B)	Test	Training	dm(A)	dm(B)	Test
nug20	54.2	103.8	nug15/25/30	bur26a	15.1	**274.9**	bur26b/26c
rou15	68.9	69.2	rou20	kra30a	49.2	**150.0**	kra30b/32
sko42	52.0	108.5	sko49/56	scr12	**257.4**	57.1	scr20
tai30a	58.0	63.2	tai25a	ste36a	55.6	**400.3**	ste36b/36c
tai35a	61.6	61.6	tai40a	tai30b	85.2	**323.9**	tai25b
tai50a	60.7	62.2	tai60a	tai35b	78.7	**309.6**	tai40b
wil50	54.2	66.7	wil100	tai50b	73.4	**313.9**	tai60b

Table 7.4: Training and Test Instances for QAP experiments (Classified).
Observe the **bold entries** in Type B training instances that show the high dominance.

Search Trajectory Analysis and Hypothesis of Better Walks

We now use the FLST visualization to come up with some hypothesis for better SLS algorithm walks. The sketch of Ro-TS-I behaviors on QAP type A and B instances are shown with **red solid lines** and our hypotheses on *better* search trajectories are shown with **blue dashed lines** in the same Figure 7.8 on tai30a and tai30b. Note that since the visualization error (`errAP`) is quite high, two APs that are drawn near (/far) each other may not be near (/far) in the actual n-dimensional space. We also expect many 'blank periods' in the visualization when the search trajectory is far from known APs.

[11] Note that taixxa and taixxb instances have the same author but they belong to two different types.

In Table 7.5 column Ro-TS-I, we observe that Ro-TS-I already has reasonably good performance on type A instances. This may be because the quality gap between local optima and BK solution is small due to the smooth fitness landscape. In Figure 7.9 label (I1-I3), we observe that Ro-TS-I trajectory does not show any obvious sign of being stuck in a local optimum. The **red circle** that denotes Ro-TS-I position appears in different APs in different iterations: visit **Medium/△** AP in bottom left screen (label I1), disappear from the FLST visualization as it is far from known APs (label I2), and visit **Bad/□** AP in top right screen (label I3).

Type A instance is an easier problem than type B as fitness landscape is smoother, i.e. most SLS runs will find solution with quality close to the BK OV. But a smooth landscape makes it hard for the SLS algorithm to decide where to navigate as 'everything' looks good. Diversifying too much may be ineffective since the SLS trajectory will likely end up in another region with similar quality. Our hypothesis: increase intensification in a region where the SLS trajectory is currently in to avoid missing the best solution in that region. Figure 7.8 (left, tai30a, middle) shows our aim to have a trajectory **(blue dashed lines)** that finds good local optima rather than Ro-TS-I trajectory **(red solid lines)** that misses them.

On the other hand, the performance of *the same* Ro-TS-I on type B instances is poor as seen in Table 7.7 column Ro-TS-I, especially for tai30b, tai35b, and tai50b. In Figure 7.11 label (I1-I3), we observe that Ro-TS-I is stuck around a **VeryBad/.** AP which is $\gg$ 3%-off BK OV. The Ro-TS-I trajectory (**red circle**) enters a region near one of this **VeryBad/.** APs at the start of search (label I1), is still there in the middle (label I2) and end of the search (label I3). Animation shows that this **VeryBad/.** AP is occasionally revisited throughout Ro-TS-I runs. Since the quality of the solutions in that region is poor, the reported best found solution is also poor.

The QAP type B fitness landscape is more rugged, i.e. the **Good/O** local optima are deeper and spread out. We hypothesize that within the limited iteration bound of $5*n^2$, rather than attempting to escape deep local optima with its own strength (e.g. via the tabu mechanism), it is better for Ro-TS-I to perform frequent strong diversifications. Figure 7.8 (right, tai30b, middle) shows that the desired trajectory **(blue dashed lines)** only makes short runs in a region before jumping elsewhere rather than Ro-TS-I trajectory **(red solid lines)** which struggles to escape a deep local optimum.

7.3.5 Ro-TS-A for QAP

White-Box Step: Intensification for Smooth Fitness Landscape

We believe that Ro-TS-I performance on type A instances can be improved. In Figure 7.10, we observe that a *single* Ro-TS-I run in $5*n^2$ iterations does not visit many good APs that are collected from several runs. How to make Ro-TS-I visit them more?

We come up with this idea to improve Ro-TS-I performance. During short runs, there may be some Ro-TS-I moves which lead to **Good/O** APs that are under tabu status and are not overridden by the aspiration criteria.

The idea for robustness by [137] used in Ro-TS-I is to randomly change Tabu Tenure during the search within a defined Tabu Tenure Range (TTR) at every **TOL**$\%*n$ steps. The TTR is defined as the interval [**TTL** ... (**TTL** + **TTD**)] which has two parameters: Tabu Tenure Low (**TTL**) and Tabu Tenure Delta (**TTD**).

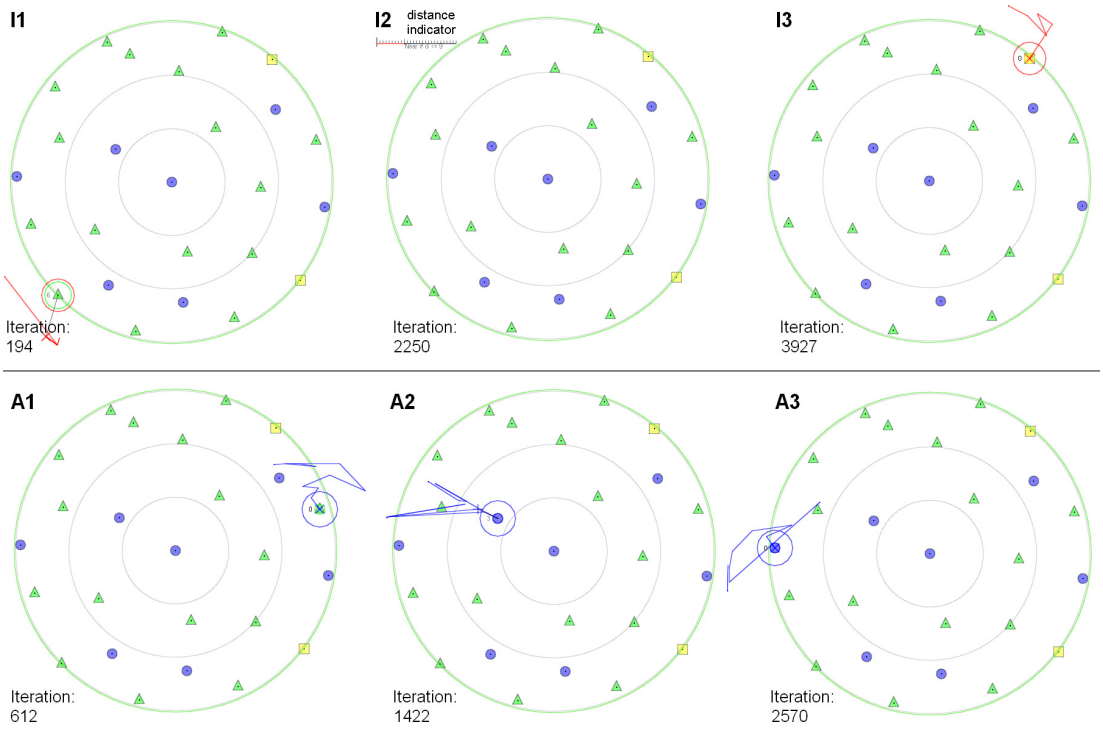


Figure 7.9: Search Trajectory of **Ro-TS-I (top)** vs **Ro-TS-A (bottom)** on tai30a

To encourage Ro-TS-I to do more intensification, we *decrease* its TTR from the settings in [137]: [90 ... 110] into a lower range and changing the robust Tabu Tenure value more often – after n steps (**TOL** = 100), not $2n$ steps (**TOL** = 200 in Figure 7.7).

Black-Box Step: Fine-Tuning

We do not know the value of **TOL** and TTR parameters for Ro-TS-I, except that both should be lower. We use black-box tuning (full factorial design) on **TOL** = $\{100, 200\}$, **TTL** = $\{40, 60, 80\}$, and **TTD** = $\{20, 30, 40\}$ and obtain **TOL** = 100 and TTR = $[40 \ldots 60]\% * n$ (**TTL** = 40, **TTD** = 20) as the TTR that works best on the training instances. We call Ro-TS-I with this tuned configuration as **Ro-TS-A**.

Training Instances

Instance	**Iters**	*BK* **OV**	**Ro-TS-I**		**Ro-TS-A**	
			$\bar{x}$	σ	$\bar{x}$	σ
nug20	2000	2507	0.15	0.27	**0.07**	0.16
rou15	1125	354210	0.21	0.45	**0.03**	0.15
sko42	8820	15812	0.17	0.11	**0.08**	0.10
tai30a	4500	1818146	0.92	0.22	**0.85**	0.33
tai35a	6125	2422002	1.13	0.39	**0.96**	0.31
tai50a	12500	4938796	1.72	0.13	**1.51**	0.15
wil50	12500	48816	**0.13**	0.04	**0.13**	0.21

Table 7.5: Ro-TS-I/A Results on Type A Training Instances (20 replications)

Table 7.5 shows that Ro-TS-A slightly outperforms Ro-TS-I. **Bold entries** that shows the best result for a certain row are mostly in the Ro-TS-A column. Wilcoxon signed-ranks test detects a significant difference between the average OV found by Ro-TS-I and Ro-TS-A on QAP type A instances (6 different pairs, $p < 0.05, T = 0.0, V = 0$).

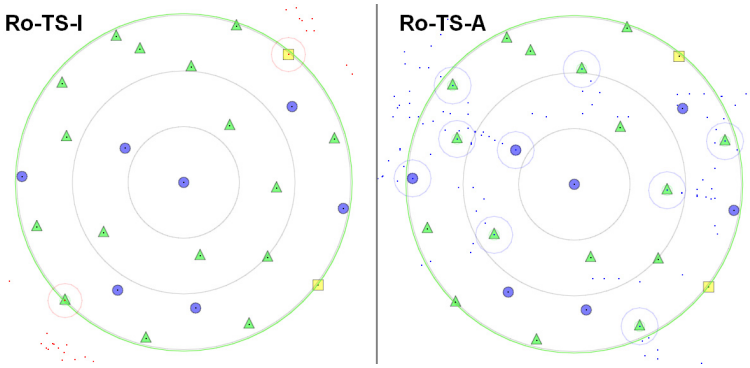


Figure 7.10: Search Coverage of **Ro-TS-I** vs **Ro-TS-A** on tai30a

In Figure 7.9 label (A1-A3), we see that the smaller TTR is still enough to make Ro-TS-A avoid solution cycling: **blue circle** that denotes Ro-TS-A position appears in different *AP*s that are located far away. This may be because it is quite easy to escape from local optima of the smooth fitness landscape of type A instances. More importantly, we observe in Figure 7.10 that the search coverage of the fine-tuned Ro-TS-A is wider than Ro-TS-I with more **Good/O** and **Medium/△** *AP*s are visited.

7.3.6 Ro-TS-B for QAP

White-Box Step: Diversification for Rugged Fitness Landscape

In Figure 7.11 label (I1-I3), Ro-TS-I is shown to be stuck around a **VeryBad/.** *AP* and does not visit the better quality *AP*s. This leads to a relatively poor performance (see Table 7.7 column Ro-TS-I and the OV visualization of Ro-TS-I in Figure 7.11 (top) which looks like a flat line when zoomed-out). With the understanding that the fitness landscape of type B instances is rugged, we suspect that the inability of Ro-TS-I to escape those *AP*s is because the *AP*s are part of deep local optima regions.

To alleviate this situation, we add a strong diversification strategy into Ro-TS-I. We consider Ro-TS-I to be stuck in a local optimum after **TOL**$\% * n$ non-improving moves. To escape, we employ a *strong* diversification mechanism called RuinAndRecreate based on the idea from [95]. This diversification strategy preserves $max(0, n - X)$ items and randomly shuffles the assignment of the other $min(n, X)$ items in the current solution. The value of X should be sufficiently large to bring Ro-TS-I out from deep local optima, but not equal to n, otherwise it will be tantamount to random restart. The rationale for this *strong* diversification heuristic is that we see in the fitness landscape that **Good/O** *AP*s in type B instances are located quite far apart but usually *not* as far as the maximum distance n (see Figure 7.11 label B3, where the highlighted distances between **Good/O** *AP*s are 15, 19, and 22, which are not close to $n = 30$).

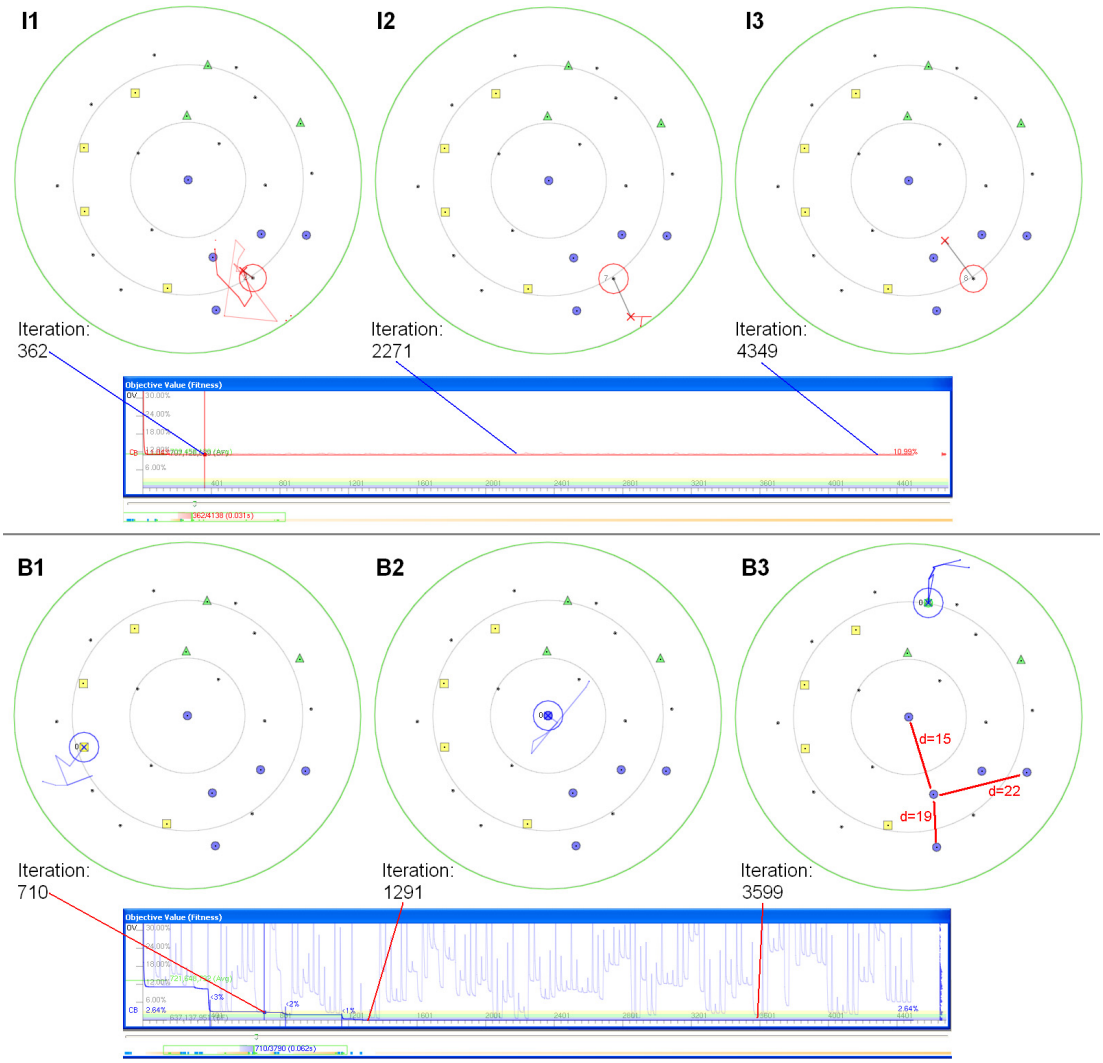


Figure 7.11: Search Trajectory of **Ro-TS-I (top)** vs **Ro-TS-B (bottom)** on tai30b

Black-Box Step: Fine-Tuning

Black-box tuning algorithm is used to determine the diversification strength[12] X. As the average instance size of our type B training instances is 31, we try X values in the middle of $[0 \ldots 30]$, i.e. $X = \{8, 10, \ldots, 22\}$. Other parameters are set to be similar as Ro-TS-A, i.e. **TOL** $= 100$, **TTL** $= 40$, **TTD** $= 40$.

The result of fine tuning using **all** type B training instances is shown in Figure 7.12.A, with the best $X = 16$. Table 7.7 column $X = 16$ shows the results on type B training instances when X is fixed. The results are already much better than Ro-TS-I results in the same Table 7.7. We call this variant as FIXED-DIVERSIFICATION strategy.

However, if we split the result w.r.t individual type B training instances as in Figure 7.12.B, we observe that the best value of X differs across different instances. For smaller instances, the best X tends to be smaller, and vice versa. The best X for each instance is around half of instance size n but not always the case.

[12]Note that we purposely show the development process using IWBBA in this section. Rather than directly give the final SLS algorithm: Ro-TS-B where we set X to be a robust value, we first show the poorer algorithm: `Fixed-Diversification` where we set X to be a fixed value.

If we split the training instances into 3 sets: small {scr12, bur26a}, medium {kra30a, tai30a, tai35b, ste36a}, and large {tai50b}, then the best X values reported are 10, 12, and 28, respectively (Figure 7.12.C). Table 7.6 shows that FIXED-DIVERSIFICATION strategy suffers from over-fitting: selected X varies depending on the given instances.

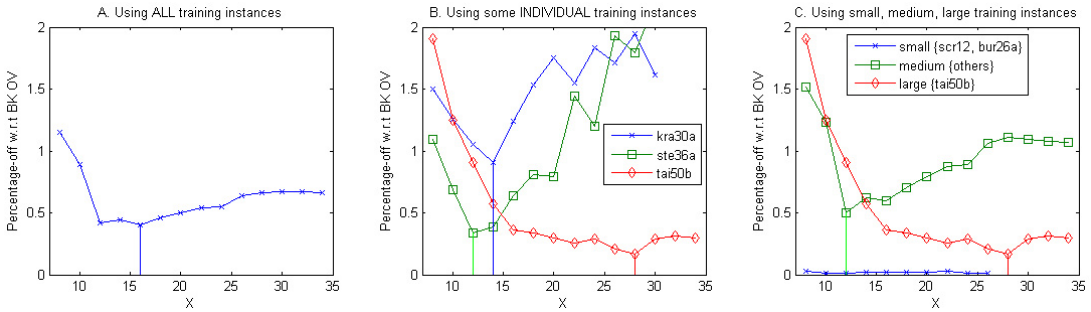


Figure 7.12: Finding the best X on type B training instances.

			Training Instances							
Instance	**Iters**	***BK* OV**	$X = 10$		$X = 12$		$X = 16$		$X = 28$	
			$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ
scr12	720	31410	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00	**0.00**	0.00
bur26a	3380	5426670	**0.02**	0.04	**0.02**	0.04	0.04	0.04	**0.02**	0.03
kra30a	4500	88900	1.26	0.62	**1.05**	0.63	1.24	0.49	1.95	0.81
tai30b	4500	637117113	1.41	1.07	**0.23**	0.32	0.27	0.36	0.30	0.09
tai35b	6125	283315445	1.57	1.73	0.40	0.34	**0.24**	0.12	0.42	0.20
ste36a	6480	9526	0.69	0.76	**0.34**	0.30	0.64	0.35	1.79	1.02
tai50b	12500	458821517	1.25	1.05	0.91	0.65	0.36	0.26	**0.17**	0.15

Table 7.6: Setting $X = 10/X = 12/X = 16/X = 28$ on Type B Training Instances (20 replications per run). Instances are sorted by size.

White-Box Step: One More Insight

To obtain better results across different instances and avoid the over-fitting issue, we conclude that X should not be *fixed* for all instances but rather be *robust* within a range correlated with the instance size n, i.e. $X = [\mathbf{XL} \ldots (\mathbf{XL} + \mathbf{XD})]\%*\text{n}$. X is randomly changed within this range after each diversification step – a 'double' Robust Tabu Search by allowing Tabu Tenure TT and diversification strength X to vary within some predetermined range. This helps maintaining the consistency of the performance quality across various QAP type B instances, especially if instance size varies.

Black-Box Step: Fine-Tuning Again

As FIXED-DIVERSIFICATION strategy above yields reasonably good results when X is set around half of the instance size n, we run the black-box fine tuning procedure (full factorial design) on a reasonable range $\mathbf{XL} = \{40, 45, 50, 55, 60\}$ and $\mathbf{XD} = \{0, 5, 10, 15, 20\}$. Again, all other parameters are the same as with Ro-TS-A, i.e. $\mathbf{TOL} = 100$, $\mathbf{TTL} = 40$, $\mathbf{TTD} = 40$. We arrived at a good range for $X = [50 \ldots 55]\% * n$ ($\mathbf{XL} = 50$, $\mathbf{XD} = 5$) that works best on the type B training instances.

We call the revised strategy as **Ro-TS-B**. Although the overall performance of Ro-TS-B on type B instances is slightly better than FIXED-DIVERSIFICATION with $X = 16$, it is not statistically significant: compare Table 7.7 column $X = 16$ with column Ro-TS-B. However, we already found that FIXED-DIVERSIFICATION tends to overfit to training instances, thus we prefer the Ro-TS-B.

Search trajectory animation of Ro-TS-B in Figure 7.11 label (B1-B3) shows that Ro-TS-B employs frequent **strong diversifications**: after visiting an AP briefly, Ro-TS-B trajectory (**blue circle**) disappears from the FLST visualization as it explores region far from known APs, then it briefly appears in another AP far from the earlier ones, and then it disappears again. This is the intended behavior of the RUINANDRECREATE strategy. Although not all APs visited by Ro-TS-B have **Good/O** quality, some do. This explains why Ro-TS-B has better performance than Ro-TS-I. The OV visualization of Ro-TS-B in the same Figure 7.11 also confirms this observation.

Figure 7.13 shows that the search coverage of Ro-TS-B (covers many APs) is much superior than Ro-TS-I (stuck in a **VeryBad/.** AP).

Training Instances

Instance	**Iters**	BK **OV**	**Ro-TS-I**		$X = 16$		**Ro-TS-B**	
			$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ
bur26a	3380	5426670	0.20	0.08	0.04	0.04	**0.03**	0.04
kra30a	4500	88900	1.88	1.43	1.24	0.49	**0.86**	0.81
scr12	720	31410	0.19	0.87	**0.00**	0.00	**0.00**	0.00
ste36a	6480	9526	1.13	1.06	**0.64**	0.35	0.77	0.53
tai30b	4500	637117113	12.12	6.65	**0.27**	0.36	**0.27**	0.44
tai35b	6125	283315445	8.10	3.89	**0.24**	0.12	0.26	0.17
tai50b	12500	458821517	6.02	3.33	0.36	0.26	**0.21**	0.14
		Average	4.23		0.40		**0.34**	

Table 7.7: Ro-TS-I/$X = 16$/B Results on Type B Training Instances (20 replications per run). Note that column $X = 16$ is duplicated from Table 7.6.

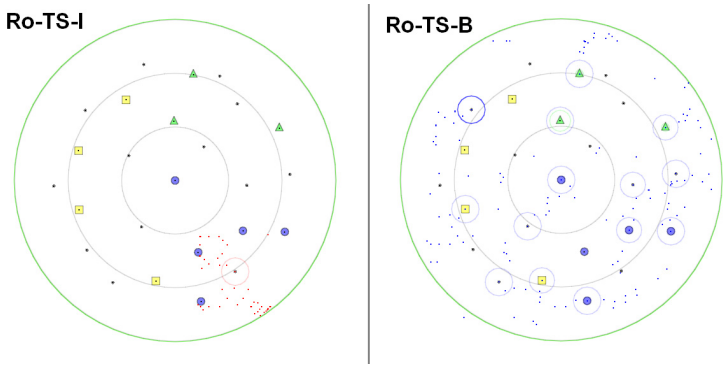


Figure 7.13: Search Coverage of **Ro-TS-I** vs **Ro-TS-B** on tai30b

7.3.7 Results on Test Instances

We compare our Ro-TS variants on the test instances using the same iteration bound: $5{*}n^2$. The results are given in Table 7.8 where a **bold entry** in a particular row/instance indicates the best result for that instance. We observe that Ro-TS-A performs *slightly*

Test Instances

Instance	Iters	*BK* OV	Ro-TS-I		Ro-TS-A		Ro-TS-B	
			$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ
Type A Test Instances								
nug15	1125	1150	0.04	0.08	0.05	0.08	**0.03**	0.07
nug25	3125	3744	0.02	0.04	**0.01**	0.04	**0.01**	0.02
nug30	4500	6124	0.23	0.18	**0.15**	0.21	0.19	0.17
rou20	2000	725522	0.24	0.16	**0.14**	0.13	0.26	0.16
sko49	12005	23386	0.22	0.08	**0.11**	0.08	0.25	0.10
sko56	15680	34458	**0.28**	0.15	0.31	0.28	0.29	0.12
tai25a	3125	1167256	1.05	0.40	**0.90**	0.38	1.28	0.29
tai40a	8000	3139370	1.23	0.29	**1.05**	0.29	1.74	0.29
tai60a	18000	7205962	1.62	0.17	**1.50**	0.19	2.13	0.24
wil100	50000	273038	0.20	0.08	0.20	0.12	**0.16**	0.04
		Average ⇒	0.51		**0.44**		0.63	
Type B Test Instances								
bur26b	3380	3817852	0.39	0.24	0.44	0.25	**0.04**	0.06
bur26c	3380	5426795	0.19	0.23	0.36	0.37	**0.00**	0.00
kra30b	4500	91420	0.30	0.39	0.85	0.91	**0.24**	0.22
kra32	5120	88700	**0.60**	0.84	1.53	1.32	0.79	0.67
scr20	2000	110030	0.35	0.64	0.96	1.20	**0.10**	0.22
ste36b	6480	15852	2.87	3.30	4.50	2.91	**0.53**	0.74
ste36c	6480	8239110	0.76	0.63	1.46	1.35	**0.32**	0.26
tai25b	3125	344355646	17.03	10.19	18.22	9.53	**0.13**	0.16
tai40b	8000	637250948	10.00	4.43	10.38	4.32	**0.10**	0.24
tai60b	18000	608215054	7.34	3.49	8.04	3.27	**0.15**	0.10
		Average ⇒	3.98		4.67		**0.24**	

Table 7.8: Ro-TS-I/A/B Results on Test Instances (20 replications per run)

better than Ro-TS-I on type A instances. Since the fitness landscapes of type A instances are smoother, any improvement will be small. Wilcoxon signed-ranks test detects that there is a significant difference between Ro-TS-A and Ro-TS-I performance (9 different pairs, $p < 0.05, T = 4.5, V = 8$). We also observe that Ro-TS-B is *much better* than Ro-TS-I on type B instances (10 pairs, $p < 0.05, T = 2.5, V = 10$).

To check the specialization of our Ro-TS variants to the problem type, we apply both Ro-TS-A or Ro-TS-B to its *opposite* instance class. We see that on type B instances, this gives worse results (underlined). Without strong diversification, the lower tabu tenure range TTR in Ro-TS-A causes it to be more stuck in deep local optima regions than Ro-TS-I. On the other hand, Ro-TS-B that jumps around the fitness landscape of type A instances mostly performs poorer than Ro-TS-A (also underlined, except for 'nug15', 'sko56', and 'wil100' instances). This shows that we have successfully tailored the SLS algorithm to match different fitness landscapes of these instances.

In summary, we have shown in this section that the FLST visualization can intuitively show 2 different fitness landscape structures in QAP: spread-smooth (type A) and spread-rugged (type B). The insights on how Robust Tabu Search (Ro-TS) [137, 138] works on these two fitness landscape types led us to design two specialized Ro-TS variants given a restricted iteration bound. Enhanced with black-box fine tuning, these two Ro-TS variants works well on test instances.

7.4 Low Autocorrelation Binary Sequence Problem

These results have been published in [58]. Note that we use several machines in this section for comparisons of runtimes.

7.4.1 Experiment Objectives

This time, we apply IWBBA using Viz on a *non classical* COP. We choose the LABS Problem (LABSP) for its difficulty: it has $O(2^n)$ search space; the current best exact algorithm [87, 88] for LABSP needs exponential time $O(1.85^n)$ to get optimal results up to $n \leq 60$; and it is also mentioned in [87] that LABSP poses a significant challenge to local search methods.

We show that with IWBBA, we can engineer a new state-of-the-art SLS for the LABSP given a baseline algorithm (**TSv0** Tabu Search algorithm by [38]).

7.4.2 Formal Problem Description

The Low Autocorrelation Binary Sequence Problem (LABSP) is a computationally difficult problem even for small instance size n. LABSP has a simple formulation: find a binary sequence $s = \{s_0, s_1, \ldots, s_{n-1}\}, s_i \in \{-1, 1\}$ of length n that minimizes the objective function[13] $g(s) = E(s)$ (which is the *quadratic* sum of the autocorrelation function C_k), or equivalently, maximizes the merit factor $F(s)$:

$$C_k(s) = \sum_{i=0}^{n-k-1} s_i s_{i+k} \qquad E(s) = \sum_{k=1}^{n-1} (C_k(s))^2 \qquad F(s) = \frac{n^2}{2E(s)} \tag{7.5}$$

Example

Let $n = 3$. We have 2^3 solutions but we only have 2 canonical ones due to symmetries:

1). $\{1, 1, -1\}$ or $\{-1, -1, 1\}$ or '21' in Run length notation[14].
$g(s) = E(s) = (C_1)^2 + (C_2)^2 = (1 * 1 + 1 * -1)^2 + (1 * -1)^2 = 0^2 + (-1)^2 = 0 + 1 = 1$
$F(s) = 3^2/(2 * 1) = 9/2 = 4.5$
The solution $\{1, 1, -1\}$ is symmetrical to $\{-1, -1, 1\}$, $\{-1, 1, 1\}$, and $\{1, -1, -1\}$.

2). $\{1, 1, 1\}$ or $\{-1, -1, -1\}$ or '3' in Run length notation.
$g(s) = E(s) = (C_1)^2 + (C_2)^2 = (1 * 1 + 1 * 1)^2 + (1 * 1)^2 = 2^2 + 1^2 = 4 + 1 = 5$
$F(s) = 3^2/(2 * 5) = 0.9$
The solution $\{1, 1, 1\}$ is symmetrical to $\{-1, -1, -1\}$, $\{1, -1, 1\}$, and $\{-1, 1, -1\}$.

7.4.3 Experimental Setup

Benchmark Instances

LABSP instances are only characterized by their length n. LABSP instances with n between $[21 \ldots 39]$ and $[40 \ldots 60]$ are used as training and test instances, respectively.

[13] In this thesis, we adopt $g(s)$ to denote the objective function. However, in the literature of LABSP, the objective function is usually denoted by $E(s)$.

[14] Each digit in Run length notation indicates the number of consecutive elements with the same sign.

FLST Quality Measures

The quality gaps between LABSP solutions are larger than TSP/QAP solutions in terms of percentage-off from BK OV. If we use the same setting, we will only see two data classes: good (the global optima) and bad (all local optima). Thus, we define $\Rightarrow$ **Good/O**: $[0-20)$, **Medium/△**: $[20-40)$, **Bad/□**: $[40-60)$, and **VeryBad/.**: $[\geq 60)$.

The most appropriate distance function to measure the distance between LABSP solutions is the Hamming distance as with QAP. For example, $d(\{1,1,-1\},\{1,1,-1\}) = 0$ and $d(\{1,1,-1\},\{-1,-1,-1\}) = 2$.

Baseline Algorithm

Although LABSP is said to be hard for local search methods [87], [38] shows a surprisingly simple yet successful Tabu Search (TS) algorithm for LABSP. We implement this algorithm according to our understanding of [38] and call it **TSv1**. The pseudo-code and the configurable parts of **TSv1** are shown in Figure 7.14.

```
TSv1(n)
 1  startTime ← clock() // remember start time
 2  CurS ← BF ← InitialSolution(n)
 3  c ← 0 // non improving moves counter
 4  while g(BF) > g(BK) // BK = Global Optima (GO) for n ≤ 60
 5  do NeighborOfCurS ← Neighborhood(CurS, n, TabuTable, AspirationCriteria(ASP))
 6     CurS ← best(NeighborOfCurS, TIE)
 7     Update TabuTable by TT% * n
 8     if g(CurS) < g(BF)
 9        then c ← 0 // reset
10             BF ← CurS
11        else if c > MS // saturated?
12                then c ← 0
13                     CurS ← InitialSolution(n) // restart
14                else c ← c + 1
15  return clock() − startTime // this algorithm reports runtime to reach BK
```

Configurable Part	Initial Choice	Remark
TT (TabuTenure)	$0.2n$	n in **TSv0** (elaborated below)
ASP (AspirationUsed)	1	When **ASP** = 1 (on), the AspirationCriteria is used
TIE (TieBreaking)	1	When **TIE** = 1 (on), the tie breaker strategy is used
MS (MaxStable)	1000	Restart criteria
InitialSolution	Random	Randomly generate bit string of length n
Neighborhood	1-bit flip	$O(n)$ move
g (ObjectiveFunction)	delta	$O(n)$ [46] instead of $O(n^2)$ computation in **TSv0**
TabuTable	bit i	This bit i cannot be flipped for **TT** steps
AspirationCriteria	Better	Override tabu if move leads to a better solution
SearchStrategy	Restart	After **MS** number of non improving moves

Figure 7.14: Code and initial configuration of **TSv1** for LABSP

TSv1 starts from a random bit string of $\{1,-1\}$ with length n. Then in line 5, it iteratively flips one bit from the current solution if that bit is not forbidden by the TabuTable (or forbidden but aspired). Note that to use AspirationCriteria, **ASP** must be on. If **TIE** is on, **TSv1** will choose one random best neighbor, otherwise the first best neighbor is always selected (line 6). Then, **TT** dictates how long a recently flipped

bit is forbidden to be flipped again (line 7). Finally, **TSv1** will restart from a random bit string again once **MS** iterations have elapsed without any improvement (line 11-14). **TSv1** stops when $g(BF) = g(BK)$ for LABSP with known optimal solution ($3 \leq n \leq 60$). Only for the runs on larger instances shown in Table 7.12, this terminating criteria (line 4) is changed to 'runtime limit' (described later in Section 7.4.6).

After implementing **TSv1**, we actually obtained the TS source code from the authors of [38]. We call the original implementation as **TSv0**.

When we benchmarked **TSv0** on our machine, a *2 GHz Core2 Duo* (see the scattered **black O** around **magenta line** in Figure 7.20), we observed that our machine produced similar performance to the *3 GHz P4 PC* used in [38].

However, our **TSv1** implementation is already **much faster** than **TSv0** (see the **red line with** $\triangle$ in Figure 7.20).

The speed difference is clear as **TSv1** terminates (reach GO for $n \leq 60$) much faster than **TSv0**. Source codes analysis reveals the following two major differences.

First, while both codes use a form of "incremental computation" to speed up the naïve $O(n^2)$ $E(s)$ computation, the actual sub-algorithms turn out to be different. Since this part is not described in [38], we implemented **TSv1** with the incremental $O(n)$ *ValueFlip* technique used by MA_{TS} [46]. It turns out that although there is some incremental calculation in **TSv0**, the computation of $E(s)$ is still $O(n^2)$.

Second, although both codes use an $O(1)$ TabuTable mechanism, they have different **TT** settings. We know that **TT** cannot be $\approx n$ as it will quickly forbid (almost) all 1-bit flip moves. Black-box tuning on several constant values $[0.1, 0.2, 0.3]n$ on some training instances helps us to set small **TT** $= 0.2n$ for **TSv1**. But, **TSv0** uses **TT** $= n$. Thus, **TSv0** does more frequent random restarts (every n+1 iterations) than the pre-determined **MS** parameter as no more valid 1-bit flip moves are available when all n bits are tabu. In [38], the authors had intended that **MS** $= 1000$ non improving iterations. However, Figure 7.15 shows that on instance $n = 27$, **TSv0** restarts every n+1 $= 28$ iterations. This is a 'failure mode' [153][15].

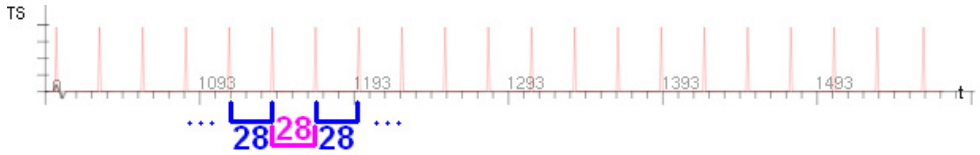


Figure 7.15: **TSv0** 'failure mode'. This Algorithm-Specific visualization in Viz shows a spike when random restart is called inside **TSv0**. Instance shown is $n = 27$.

We can see in Figure 7.20 that our **TSv1** runtimes on *2 GHz Core2 Duo* are *already* comparable to the recent state-of-the-art MA_{TS} [46]. When run on a *3 GHz P4 PC*, **TSv1** runtimes is already [1.7–5.6] times faster than MA_{TS} for LABSP $40 \leq n \leq 55$. This *3 GHz P4 PC* is probably just 1.25 times faster than the *2.4 GHz P4 PC* used in [46]. We remark that this shows that the random restart strategy in **TSv1**/**TSv0** is actually effective and it performs better than the benchmarking in [46] would indicate.

[15] This shows that an SLS can be 'buggy' and yet it still manage to obtain reasonably good results.

7.4.4 Fitness Landscape Search Trajectory Analysis

Fitness Landscape Analysis: LABSP Fitness Landscape is Hard for SLS

Previous researchers, e.g. [30, 87, 46] have shown several features of LABSP fitness landscape. In Table 7.9, we show some basic LABSP fitness landscape statistics to re-confirm previous findings. We perform an exact enumeration of all 2^n LABSP solutions (all solutions are feasible as LABSP is unconstrained) for small instances up to $n = 24$.

n	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
$\|GO\|$	4	8	4	28	4	16	24	40	4	16	4	72	8	32	44	16	8	8	4	24	24	8
level	2	3	4	4	9	13	19	17	33	50	60	46	96	117	139	100	203	254	295	201	405	470

Table 7.9: Statistics of Small LABSP Instances $n \leq 24$

Each LABSP instance n has *several* Global Optima (GO) (≥ 4, because of symmetry). However, the number of GO ($|GO|$) does not scale with n. The $|GO|$ for $3 \leq n \leq 24$ is shown[16] in Table 7.9 row $|GO|$. The second row **level** shows the number of different Objective Values (OVs) of the 2^n solutions. These values are also much less than 2^n which implies that an OV level is likely shared by *many* different LABSP solutions.

For the other LABSP training instances $25 \leq n \leq 39$, exact enumeration already takes much more than 1 minute, but SLS algorithms can consistently hit the GO in ≤ 1 second in our machine. In order to get more insights about LABSP fitness landscape, we use the FLST visualization. We run **TSv1** to sample diverse and high quality Local Optima (LO) from the fitness landscape of these LABSP instances. Our sampling strategy exploits the symmetries in LABSP: when **TSv1** reaches a solution with OV equals with the known optimal OV (a GO) for that particular LABSP instance, we can immediately generate all the symmetries[17] of this GO solution. This sampling strategy is used to get a clearer picture of the LABSP fitness landscape.

In Figure 7.16.A1 & A2, we see that without utilizing symmetry in GO/LO sampling, we are not immediately aware of the existence of other GO and the *Hamming distance* from current LO to the nearest GO found seems to be large, i.e. $> n/2$, which is $> 27/2 = 13$ in this case. Observe the distance (**red lines**) from the highlighted LO (**yellow rectangle** with **orange circle**) to the two GO found (**BlueCircle O**) are $\{15, 17\}$ in A1 and $\{15, 19\}$ in A2. All distances of the **red lines** are > 13.

By exploiting symmetry when sampling the GO/LO, all 4 GO[18] of instance $n = 27$ are also 'found' when **TSv1** hits *any* GO. In Figure 7.16.B1 & B2, we see that the positions of GO are spread out. This suggests that wherever the current solution is, it should be nearer to one GO (the *nearest GO*) than to other GOs. Further observations reveal that the distance between LO to the nearest GO is usually *not too close* but bounded by $n/2$. In the same figure, we observe that the distance (**blue lines**) between the highlighted LO to the *nearest* GO are both 8. This is $\gg 0$ and $\leq n/2$.

[16] [21] has provided the list of $|GO|$ for $3 \leq n \leq 64$. However, some of which are approximate numbers.
[17] Symmetry is problem-specific. For LABSP, reversing, complementing all/even-only/odd-only bits, and combination of these operators can be used to generate the symmetries of a LABSP solution.
[18] There can be more than 4 symmetries in a LABSP instance. For $n = 27$, there are only 4 GO.

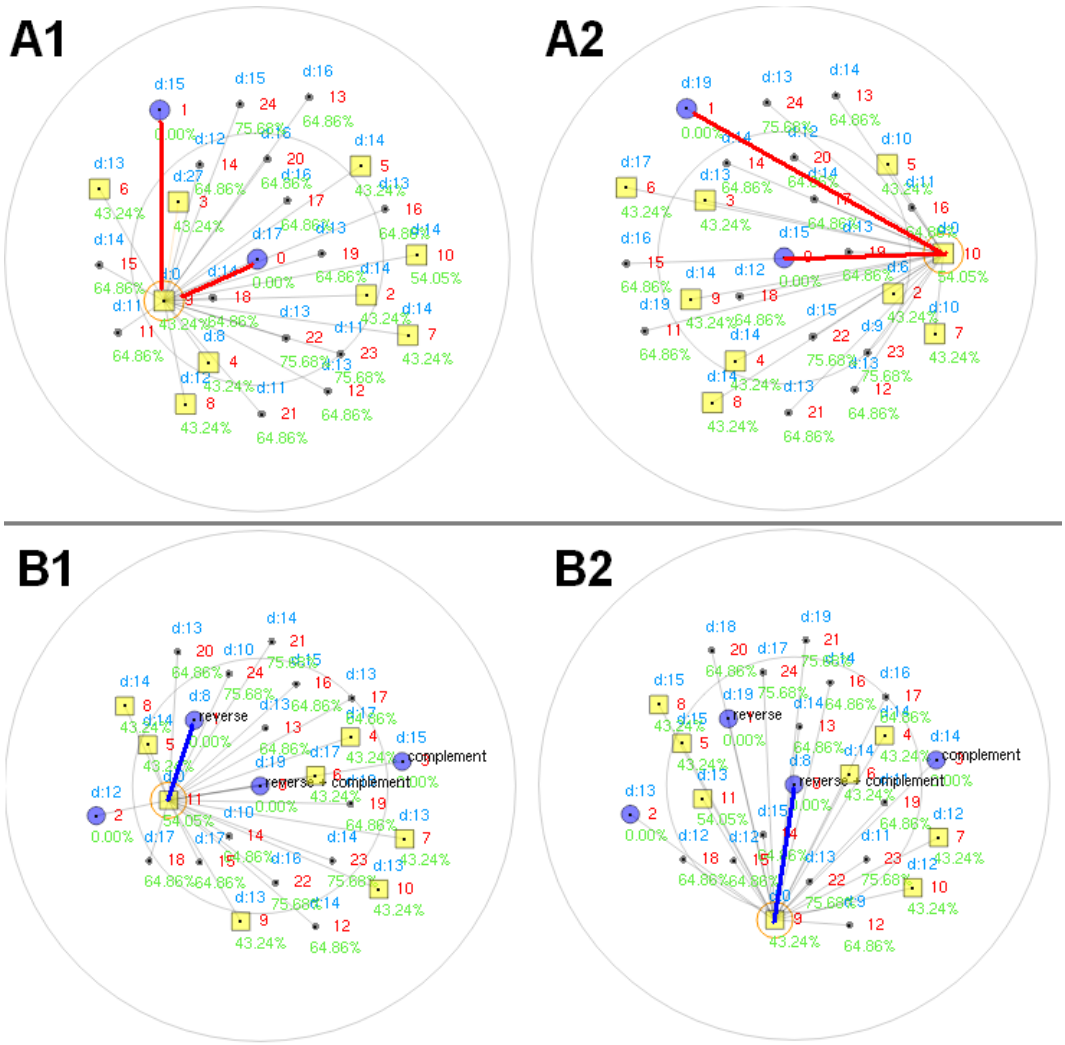


Figure 7.16: FLST visualization for LABSP $n = 27$ (FLO mode) with 4 GO (**blue circles**). Row A *does not exploit symmetry* while row B *exploits symmetry*.

The *DiversityIndex* of LABSP instance $n = 27$ in Figure 7.16.B1 & B2 is medium: 0.51. This causes medium visualization error `errAP` in Figure 7.16.B1 & B2. The LABSP FLST visualization is not as precise as in TSP FLST visualizations, but not as bad as in QAP FLST visualizations. To avoid misunderstandings, we use the distance highlights (focus on one AP and show the distances to other APs) as in Figure 7.16.

By using exact enumeration for small LABSP instances $3 \leq n \leq 24$ (see Table 7.10), we have checked that on average 95.93% of the 2nd best solution (which is an LO) have Hamming distance around $[n/5 \ldots 2n/5]$ bits away from the nearest GO. For example, for $n = 17$, there are 24 2nd best LO with distance 2 from nearest GO – denoted as 2 (24), and also: 3 (8), 4 (56), and 5 (16). Out of these $24 + 8 + 56 + 16 = 104$ 2nd best LO, around $8 + 56 + 16 = 80$ of them (76.92%) are within $[17/5 = 3 \ldots 2 * 17/5 = 6]$ distance units away from their nearest GO (indicated with **bold entries**).

In summary: The several GO of a LABSP instance are spread like 'golf holes' (isolated) in irregular LABSP fitness landscape. This causes difficulties for standard SLS algorithms to work well especially with large n [87]. However, we have identified that most LO are within $[n/5 \ldots 2n/5]$ bits away from their nearest GO.

n	$[\frac{n}{5} \ldots \frac{2n}{5}]$	d(2nd, nearest-GO) (frequency)	$\lvert 2nd \rvert$	%
3	[0 . . . 1]	**1 (4)**	4	100
4	[0 . . . 1]	**1 (4)**	4	100
5	[1 . . . 1]	**2 (12)**	12	100
6	[1 . . . 2]	**1 (24)**	24	100
7	[1 . . . 2]	**1 (8), 2 (16)**	24	100
8	[1 . . . 3]	**1 (40), 3 (8)**	48	100
9	[1 . . . 3]	**2 (80)**	80	100
10	[2 . . . 4]	1 (48), **2 (104), 3 (16)**	168	71.43
11	[2 . . . 4]	**2 (4), 3 (8), 4 (12)**	24	100
12	[2 . . . 4]	**3 (16)**	16	100
13	[2 . . . 5]	**4 (16), 5 (8)**	24	100
14	[2 . . . 5]	1 (32), **2 (56), 3 (144), 4 (64)**	296	89.19
15	[3 . . . 6]	**3 (24), 4 (24), 5 (56)**, 7 (8)	112	92.86
16	[3 . . . 6]	**3 (16), 5 (24)**	40	100
17	[3 . . . 6]	2 (24), **3 (8), 4 (56), 5 (16)**	104	76.92
18	[3 . . . 7]	1 (8), **4 (24), 5 (56), 6 (32)**	120	93.33
19	[3 . . . 7]	**6 (4), 7 (12)**	16	100
20	[4 . . . 8]	**6 (8), 8 (24)**	32	100
21	[4 . . . 8]	**6 (8), 8 (8)**	16	100
22	[4 . . . 8]	1 (8), **4 (24), 5 (48), 6 (64), 7 (56), 8 (16)**, 9 (24)	240	86.67
23	[4 . . . 9]	**5 (16), 6 (16), 7 (68), 8 (56), 9 (20)**	176	100
24	[4 . . . 9]	**8 (8)**	8	100
		Average ⇒		95.93

Table 7.10: Properties of 2nd Best Solutions w.r.t Nearest GO on LABSP $3 \leq n \leq 24$

Search Trajectory Analysis and Hypothesis of Better Walks

We want to improve over **TSv1** (and **TSv0**). We analyze **TSv1** search trajectory with the same FLST visualization. In SCO+STD mode, a circle+lines are drawn on/around the nearest sampled GO/LO if the current solution is "near" it. In this experiment, we define two solutions a and b are near when Hamming distance(a, b) $\leq 20\% * n$.

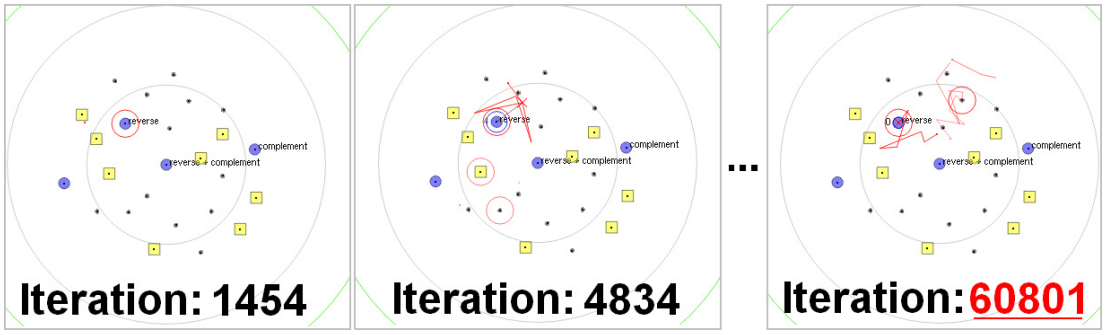


Figure 7.17: FLST visualization for LABS $n = 27$ (SCO+STD mode)

Using this feature, we observe the following behavior on almost all training instances tried: **TSv1** happens to be near a GO in the *earlier phase* of the search (see Figure 7.17, iteration 1454), but **TSv1** does not immediately navigate there. **TSv1** then wanders to another region near *another* GO, maybe due to random restart strategy, etc. Then,

it gets near the same GO again, but this time, it fails too (see Figure 7.17, iteration 4834). Only *thousands iterations later*, **TSv1** gets near to this GO again (see Figure 7.17, iteration **60801**) and this time **TSV1** manages to find the GO.

FLST visualization points out an obvious strategy: A better walk for **TSv1** is to concentrate on the *nearest GO* from the start! There are a number of (≥ 4) GO for each LABSP instance, so do not scatter the search efforts.

7.4.5 TSv7 for LABSP

White-Box Step: Utilizing the Insights

The observations and insights about the LABSP fitness landscape and current **TSv1** behavior in Section 7.4.4, plus some information from the box below, lead us to engineer a better SLS algorithm. The resulting TS variant is called **TSv7**. The pseudo-code and configurable parts of **TSv7** are shown in Figure 7.19.

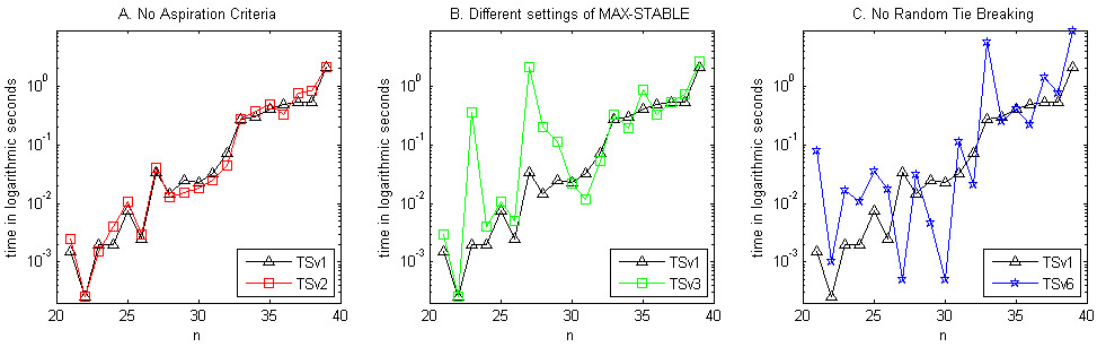


Figure 7.18: Experiments with various TS settings (**TSv1** - **TSv6**)

We have experimented with other variants of **TSv1** to gain insights on the effect of some SLS components.

In **TSv2**, we turn off the aspiration criteria, i.e. by setting parameter **ASP** = 0. It Figure 7.18.A, we observe that there is no significant difference whether we use aspiration criteria or not.

In **TSv3**, we set **MS** = 1000000, which is tantamount to *not* using random restart (**TSv4** and **TSv5** variants are not shown). We observe in Figure 7.18.B, that **TSv3** performance drops (more runtime is needed to reach optimal results). This shows that random restart is an important component for **TSv1**.

In **TSv6**, we turn off tie breaking feature by setting parameter **TIE** = 0. This causes inconsistent performance across training instances as seen in Figure 7.18.C.

These experiments suggest that TS for LABSP needs frequent restarts and tie breaking, but it does not need aspiration criteria.

TSv7 given in Figure 7.19 is **TSv1** with **ASP** fixed to 'off' **(line 5)**, **TIE** fixed to 'on' **(line 6)**, and uses a different diversification strategy (line 15-20), elaborated below.

```
TSv7(n)
 1  startTime ← clock() // remember start time
 2  CurS ← BF ← InitialSolution(n)
 3  c ← k ← 0 // s = non improving moves counter, k = local restart counter
 4  while g(BF) > g(BK) // BK = GO for n ≤ 60
 5  do NeighborOfCurS ← Neighborhood(CurS, n, TabuTable) // ASP = 0
 6     CurS ← best(NeighborOfCurS, 1) // TIE = 1
 7     Update TabuTable by TT = [TTL ... (TTL + TTD)]% ∗ n
 8     if g(CurS) ≤ g(BF)
 9       then r ← BF ← CurS // r is reference solution
10            if g(CurS) < g(BF) // reset
11              then c ← k ← 0
12              else c ← c + 1
13       else if c > MSp ∗ n // saturated?
14              then c ← 0
15                   k ← k + 1
16                   if k > THR ∗ n // local restart near reference solution r
17                     then k ← 0
18                          r ← localRestart(r, PUTT% ∗ n/2)
19                          CurS ← r ← optimize(r) // to ensure r is an LO
20                     else CurS ← localRestart(r, PUTT% ∗ n)
21              else c ← c + 1
22  return clock() − startTime
```

Configurable Part	Choices	Selected Parameter Value & Remark
PUTT	[25, 33, 40]	Selected = 25
MSp (MaxStable-p)	> 0 [2, 4, 6]	Selected = 2, let **TSv7** rely more on local restart
THR (Threshold)	> 0 [2, 4, 6]	Selected = 2
TTL (TTLow)	Low [5, 10, 20]	Selected = 20
TTD (TTDelta)	Low [5, 10, 20]	Selected = 20

Figure 7.19: Code and initial configuration of **TSv7** for LABSP. The code differs with **TSv1** mainly on line 15-20. Other configurable parts are similar with **TSv1**.

Basically, **TSv7** does not do random restart if **MSp***n iterations (now parameterized w.r.t n: line 13) have elapsed without any improvement, but rather, **TSv7** does a *local restart*[19] by perturbing **PUTT**%*n random bits from a reference solution r (line 20). This r is a Local Optima (LO), which may be the 2nd best solution elaborated in Section 7.4.4. A weaker diversification like this put the **TSv7** trajectory around $[n/5 \ldots 2n/5]$ distance units away from r, where the GO is possibly located. A robust TS strategy as in QAP with [**TTL** ... (**TTL**+**TTD**])%*n, helps **TSv7** to explore this region. We keep intensifying around r until **THR***n attempts failed (line 15-16). Then, a new reference point r is selected by doing a light local restart by perturbing **PUTT**%*$n/2$ random bits from reference solution r so that **TSv7** does not navigate too far from its current position. We optimize the resulting solution with short TS run to ensure r is an LO (line 17-19). All strategies are designed to make **TSv7** search for the *nearest* GO.

Black-Box Step: Fine-Tuning

The next step is to configure **TSv7** using the black-box tuning algorithm. The choice of values and the selected configuration are indicated in Figure 7.19.

[19]The name of parameter **PUTT** comes from golf term that means: 'strike a golf ball lightly'.

7.4.6 Results on Test Instances

Figure 7.20 shows the performance of **TS** (timings from [38]), $\mathbf{MA}_{TS}$ (timings from [46]), and **TSv0**/**TSv1**/**TSv7** (run on 2 GHz Core2 Duo). Note that the y-axis uses a logarithmic scale. We see that **TSv7** strategy is better than the original random restart strategy used in **TSv0/TSv1**. The performance gap is easily noticeable on larger $n = \{50, 55, 57, 60\}$.

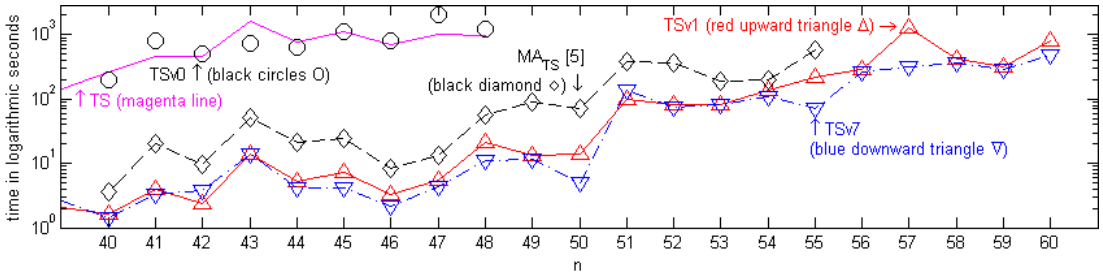


Figure 7.20: Comparison of average runtimes (20 runs) between {**TS** [38], **TSv0 O**}, {$\mathbf{MA}_{TS}$ **[46]** ◇}, and {**TSv1** △, **TSv7** ∇} for LABSP with known optimal OVs ($40 \leq n \leq 60$). The machine used for **TSv0/TSv1/TSv7** is a 2 GHz Core2 Duo.

To analyze the results, we use the Wilcoxon signed-ranks test. It detects a significant difference between the average runtimes of **TSv1** and **TSv7** on LABSP $40 \leq n \leq 60$ (21 pairs, $p < 0.05, T = 27.5, V = 67$). Since both TS variants use the same incremental OV computation and run on the same hardware, this difference in average runtimes can be attributed to the new stochastic strategy (local restarts in order to search for *nearest* GO in **TSv7** versus full random restarts in **TSv1**)[20].

Solver	Machine used	Instances	Growth Rate	**TSv7**'s Rate	Speed-up
B&B [87]	Sun UltraSparc I 170	[15-44]	1.85^n	$9.64e\text{-}7*1.43^n$	1.294^n
CLS [112]	300 Mhz DEC server	[3-48]	1.68^n	$4.78e\text{-}5*1.27^n$	1.323^n
LSR [113]	733 Mhz P3 (PC)	[7-38]	1.51^n	$4.40e\text{-}5*1.24^n$	1.218^n
ES [93, 21]	266 Mhz (P3 PC)	[20-47]	$7.11e\text{-}4*1.40^n$	$3.90e\text{-}7*1.45^n$	$1.8e3*0.966^n$
TSv0 [38]	3 Ghz (P4) PC	[21-48]	$1.69e\text{-}5*1.49^n$	$4.01e\text{-}7*1.45^n$	$4.2e1*1.028^n$
KL [21]	266 Mhz (P3 PC)	[20-47]	$1.29e\text{-}5*1.46^n$	$3.90e\text{-}7*1.45^n$	$3.3e1*1.007^n$
MA_{TS} [46]	2.4 Ghz P4 PC	[40-55]	$8.87e\text{-}5*1.32^n$	$2.88e\text{-}5*1.31^n$	$3.080*1.008^n$
TSv1	2 Ghz Core2 Duo	[40-60]	$5.03e\text{-}6*1.37^n$	$1.03e\text{-}5*1.34^n$	$0.488*1.022^n$
TSv7	2 Ghz Core2 Duo	[40-60]	$1.03e\text{-}5*1.34^n$	$1.03e\text{-}5*1.34^n$	1

Table 7.11: Overall Comparison of the Growth Rate of Various LABSP Solver

Table 7.11 gives the growth rate of **TSv7** w.r.t other existing LABSP solver (see column 'Solver' and 'Machine used'). Column 'Instances' shows the instance sizes reported in the respective paper. Some papers [87, 112, 113] already mentioned their algorithm's growth rates as in column 'Growth Rate'. For other papers, we measure the *linear least square fit* on the logarithm of the average reported runtimes. In column '**TSv7**', we perform the same linear fit on **TSv7** runtimes using the *same* instances mentioned in column 'Instances'[21]. Finally, column 'Speed-up' is the result of dividing the growth rate of the solver being compared with **TSv7**. This shows that **TSv7** is at least as good and probably better, a state-of-the-art, SLS algorithm for LABSP (as of July 2009).

[20] Turning off aspiration criteria shown in **TSv2** does not significantly affect **TSv1** performance.
[21] The result of linear fit depends on the number of given data points.

Table 7.12 explores the frontier of LABSP instances, $61 \leq n \leq 77$, where optimal values have yet to be proven. These longer runs are performed on a 2.33 GHz Core2 Duo PC. For $61 \leq n \leq 70$, we use a runtime limit based on the estimated runtime projection from Figure 7.20. For $n > 70$, we use a runtime limit of 10 hours for scalability reasons. We see that **TSv7** manages to obtain relatively good LABSP solutions (by the merit factor[22] F) in reasonable running time.

n	E(s)	F(s)	Runtime	Limit	Best Found LABSP in Run Length Notation [87]
61	226	8.23	3 m	1.1 h	3321111211123518312122111131131 1
62	235	8.18	8 m	1.5 h	1122122127111115111211431114223 21
63	207	9.59	4 m	2.0 h	2212221151211451117111112323231
64	208	9.85	47 m	2.7 h	2232241113411211151111172122122 12
65	240	8.80	2.2 h	3.7 h	1323232111117111541121511222122 11
66	265	8.22	3.1 h	4.9 h	2432112312311211212412318111111 1311
67	241	9.31	4.1 h	6.6 h	12112111211222B2221111111112224542
68	250	9.25	6.6 h	8.8 h	1111111114114723212325141211222121 2
69	274	8.69	8.2 h	11.8 h	1111111111411472321232514121122212 12
70	295	8.31	12.4 h	15.8 h	2324412117222141611252123111111 111
71	275	9.17	7.8 h	10.0 h	2412441241722221111131123112112311 21
72	300	8.64	2.4 h	10.0 h	1111114111444171151122142122224222
73	308	8.65	1.2 h	10.0 h	1111112311231122113111212114171322374
74	349	7.85	0.2 h	10.0 h	1132132161233312511141212112251113 1111
75	341	8.25	8.0 h	10.0 h	1212213212121121111113111161843321 3232
76	338	8.54	4.6 h	10.0 h	1112111122343221111341142122112213 11B11
77	366	8.10	3.9 h	10.0 h	1111111913422224311233122134112121 12112

Table 7.12: Best found LABSP solutions using **TSv7**: $61 \leq n \leq 77$.

In summary, we have shown in this section that with FLST visualization, we can visualize the irregular LABSP fitness landscape where Global Optima (GO) are isolated. With the obtained insights, we managed to improve the design of a Tabu Search algorithm (**TSv1** [38]) to search for the nearest GO. The resulting Tabu Search after fine-tuning (**TSv7**) is the state-of-the-art SLS solver for LABSP as of July 2009.

[22]Optimal LABSP solution has merit factor $F = [7 \ldots 8]$ for $3 \leq n \leq 60$ and is conjectured to be around this range for higher n [21].

Chapter 8

Conclusions

It is good to have an end to journey towards;
but it is the journey that matters, in the end.
— **Ursula K. LeGuin**

In this last chapter, we highlight the main contributions of this PhD thesis, followed by few pointers for future works.

8.1 Contributions

In this thesis, we have identified two important and related Stochastic Local Search (SLS) engineering problems: the Design and Tuning Problem (DTP) in Chapter 3. There are various white-box and black-box approaches proposed in the literature but so far no approach is clearly superior in addressing all types of the SLS DTP. To address this issue, we have made the following contributions:

Fitness Landscape Search Trajectory Visualization (Chapter 4)

- We have created a *generic* white-box Fitness Landscape Search Trajectory (FLST) visualization for analyzing the Combinatorial Optimization Problem (COP) Fitness Landscape and the SLS Trajectory behavior on it [83, 56, 59, 60, 57, 61, 58].
- As the FLST visualization attempts to visualize n-dimensional fitness landscape in 2-D, we inherently introduce visualization errors. In this thesis, we have analyzed the current limitations of the FLST visualization in Chapter 4 and provide pointers for future works in Section 8.2.

SLS Visualization Tool Viz (Chapter 5)

- We have built an SLS visualization tool. Viz [60] shows the above-mentioned FLST visualization and other white-box visualizations and statistical tools.

- We have designed the VIZ's User Interface (UI) with various information visualization techniques to empower the human user in understanding his SLS behavior.

Integrated White+Black Box Approach (Chapter 6)

- We have proposed the INTEGRATED WHITE+BLACK BOX APPROACH (IWBBA) [61]. This methodology combines the strength of both approaches: after improving the SLS design and reducing potential configuration space using white-box approaches, one can use the black-box approaches to further fine tune the SLS algorithm. This is a potential methodology to address the SLS DTP.

- We have empowered VIZ with simple black-box tuning tools to support IWBBA.

Results that Confirm Contributions (Chapter 7)

- We have successfully applied IWBBA using VIZ on three different scenarios on three different COPs. We show that with the FLST visualization, one can examine what are the characteristics of the fitness landscape of his COP and design an SLS algorithm that suits that fitness landscape:

    1. Exploiting the 'Big Valley' property in the Traveling Salesman Problem (TSP) by using Iterated Local Search (ILS) with controlled diversifications [59, 60, 61].
    2. Realizing that there are at least two different classes of instances in the Quadratic Assignment Problem (QAP): spread-smooth and spread-rugged. If SLS runs are limited to be short, it is better to do intensification on the spread-smooth instances and strong diversifications on the spread-rugged instances [57, 61].
    3. Understanding that the irregular fitness landscape of the Low Autocorrelation Binary Sequence (LABS) Problem has multiple Global Optima (GO) and good Local Optima (LO) are not too close to the nearest GO. A local restart strategy developed based on these insights is shown to be the reason that our Tabu Search algorithm (TSv7) performs at the state-of-the-art level (as of July 2009 [58]).

8.2 Future Works

Improving FLST Visualization

Although the FLST visualization can already be used to help in answering *all* the questions posed in Section 4.2 and 4.3 and it was shown to work well in Chapter 7, there are, however, some limitations to be addressed in future work (ranked by importance):

1. Scalability issue 1. The error of the FLST visualization (`errAP`) increases with $APSet$ size. The rate of `errAP` increase depends on the COP characteristics. Generally, COP instances with high $DiversityIndex$ tend to have higher `errAP`. If `errAP` is beyond an acceptable threshold, the FLST visualization becomes misleading and must be used with caution. Currently, we limit $APSet$ size to be $[25 \dots 50]$ which is suitable to analyze only small training instances.

2. Scalability issue 2. FLST visualization shows the positions of the search trajectory over time. Due to the limited attention span, it is tedious for the human user to observe a long search run which has thousands of iterations. Currently, we limit FLST analysis to short runs. An alternative method may be to select a subset of solutions $\in ST$ (e.g. the good ones, etc) to be animated in the FLST visualization.

3. The search trajectory information is not meaningful during the period when the search is exploring the fitness landscape *far* from any APs that we have, forcing the FLST visualization to show nothing (i.e. at time t, $d(s^t, AP) > \delta; \forall$ $AP \in APset$). This 'blank period' is not fruitful for SLS algorithm analysis and caused by the limited $APSet$ size. As we cannot have a large $APSet$, what other alternative information can we show to the user during this period?

4. At its current state, the FLST visualization will be too complex when more than two search trajectories are displayed at the same time. When used on population-based SLS algorithm like Genetic Algorithm (GA), we will see x points rather than one moving on the FLST visualization over time (where x is the population size). This may be too complex and further research should be conducted for better ways of visualizing population-based SLS algorithm (e.g. we may choose to only show the best individual at every generation).

5. Scalability issue 3. This approach is not suitable for solving COPs with large training sets as the algorithm designer has to run the FLST visualization over a large number of instances. Our current solution is to limit the number of training instances used in the white-box part of IWWBA.

6. As of July 2009, the FLST visualization has not been applied to any COP with infeasible regions – regions with solutions that do not satisfy the hard constraints of the COP – like Multidimensional Knapsack Problem [27], Oversubscribed Scheduling Problem [122], etc. It may be interesting to observe the performance of the SLS algorithm with the presence of infeasible regions.

7. FLST visualization is not designed to visualize SLS algorithms that dynamically alter the fitness landscape, like Guided Local Search (GLS) which keeps changing the objective function throughout the search.

8. FLST visualization is also not designed to visualize constructive SLS algorithm during the construction phase because distance information between 'partial solutions' is ill-defined. FLST visualization is still applicable for constructive SLS algorithm (e.g. ACO) as long as it is used to visualize the series of complete solutions created by the constructive SLS algorithm per iteration.

Concluding Remarks

Soli Deo Gloria

Bibliography

[1] The HuGS Platform: A Toolkit for Interactive Optimization. In *Advanced Visual Interfaces, May 2002, Trento, Italy*, 2002.

[2] Emile Aarts and Jan Karel Lenstra. *Local Search in Combinatorial Optimization*. John Wiley and Sons, 1997.

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

[4] Luis von Ahn, Manuel Blum, Nicholas J. Hopper, and John Langford. CAPTCHA: Using Hard AI Problems for Security. In *Advances in Cryptology, Eurocrypt*, pages 294–311, 2003.

[5] Ravindra K. Ahuja, Krishna C. Jha, James B. Orlin, and Dushyant Sharma. Very Large-Scale Neighborhood Search for the Quadratic Assignment Problem. Technical report, MIT Sloan School of Business, 2002.

[6] Lee Altenberg. Fitness Distance Correlation Analysis - An Instructive Counterexample. In *International Conference on Genetic Algorithm*, 1997.

[7] 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.

[8] 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.

[9] Carlos Ansótegui, Meinolf Sellmann, and Kevin Tierney. A Gender-Based Genetic Algorithm for the Automatic Configuration of Algorithms. In *Principles and Practice of Constraint Programming*, pages 142–157, 2009.

[10] Krzysztof Apt. *Principles in Constraint Programming*. Cambridge University Press, 2003.

[11] Thomas Bäck. Parallel Optimization of Evolutionary Algorithms. In *Parallel Problem Solving From Nature*, pages 418–427, 1994.

[12] Prasanna Balaprakash, Mauro Birattari, and Thomas Stützle. Improvement Strategies for the F-Race Algorithm: Sampling Design and Iterative Refinement. In *Thomas Bartz-Beielstein et. al., editors, Proceedings of Hybrid Metaheuristics*, pages 108–122, 2007.

[13] 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.

[14] Thomas Bartz-Beielstein. *Experimental Research in Evolutionary Computation: The New Experimentalism*. Springer, 2006.

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

[16] Sven Becker, Jens Gottlieb, and Thomas Stützle. Applications of Racing Algorithms: An Industrial Perspective. In *Artificial Evolution*, pages 271–283, 2006.

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

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

[19] Christian Blum and Andrea Roli. Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison. *ACM Computing Surveys*, 35:268–308, 2003.

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

[21] Franc Brglez, Yu Li Xiao, Matthias F. Stallmann, and Burkhard Militzer. Reliable Cost Predictions for Finding Optimal Solutions to LABS Problem: Evolutionary and Alternative Algorithms. In *International Workshop on Frontiers in Evolutionary Algorithms*, 2003.

[22] Rainer E. Burkard, Stefan E. Karisch, and Franz Rendl. A Quadratic Assignment Problem Library. *European Journal of Operational Research*, 55:115–119, 1991.

[23] Edmund K. Burke and Graham Kendall. *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*. Springer, 2005.

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

[25] Marco Chiarandini, Chris Fawcett, and Holger H. Hoos. A Modular Multiphase Heuristic Solver for Post Enrolment Course Timetabling. In *International Conference on the Practice and Theory of Automated Timetabling*, 2008.

[26] 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.

[27] Paul C. Chu and John E. Beasley. A Genetic Algorithm for the Multidimensional Knapsack Problem. *Journal of Heuristics*, 4:63–86, 1998.

[28] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, second edition, 2001.

[29] Teodor Gabriel Crainic, Michel Gendreau, and Louis-Martin Rousseau. Special Issue on Recent Advances in Metaheuristics. *Journal of Heuristics*, 2009.

[30] CSPLIB. A Problem Library for Constraints.
http://www.csplib.org.

[31] Giuseppe Di Battista, Peter Eades, Roberto Tamassia, and Ioannis G. Tollis. Algorithms for Drawing Graphs: an Annotated Bibliography. *Computational Geometry: Theory and Applications*, 4:235–282, 1994.

[32] Luca Di Gaspero and Andrea Schaerf. EASYLOCAL++: an object-oriented framework for the flexible design of local search algorithms and metaheuristics. In *4th Metaheuristics International Conference*, 2001.

[33] Luca DiGaspero, Andrea Roli, and Andrea Schaerf. EasyAnalyzer: An Object-Oriented Framework for the Experimental Analysis of Stochastic Local Search Algorithms. In *Engineering Stochastic Local Search Algorithms*, pages 76–90, 2007.

[34] Karl F. Doerner, Michel Gendreau, Peter Greistorfer, Walter J. Gutjahr, Richard F. Hartl, and Marc Reimann. *Metaheuristics: Progress in Complex Systems Optimization*. Springer, 2007.

[35] Grágoire Dooms, Pascal van Hentenryck, and Laurent Michel. Model-Driven Visualizations of Constraint-Based Local Search. In *Principles and Practice of Constraint Programming*, pages 271–285, 2007.

[36] Marco Dorigo and Gianni Di Caro. The Ant Colony Optimization Meta-Heuristic. In *New Ideas in Optimization*, pages 11–32. McGraw-Hill, 1999.

[37] Marco Dorigo and Thomas Stützle. *Ants Colony Optimization*. MIT Press, 2004.

[38] Iván Dotú and Pascal van Hentenryck. A Note on Low Autocorrelation Binary Sequences. In *Lecture Notes in Computer Science*, volume 4204, pages 685–689. Springer, 2006.

[39] Johann Dreo, Alain Petrowski, Patrick Siarry, and Éric D. Taillard. *Metaheuristics for Hard Optimization: Methods and Case Studies*. Springer, first edition, 2006.

[40] Ágoston Endre Eiben and Márk Jelasity. A Critical Note on Experimental Research Methodology in EC. In *IEEE International Conference on Evolutionary Computation*, pages 582–587, 2002.

[41] Mica R. Endsley. Theoretical Underpinnings of Situation Awareness: A Critical Review. In *Endsley and Garland (eds). Situation Awareness Analysis and Measurement.* Lawrence Erlbaum Associates, Mahwah, NJ., 2000.

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

[43] Stephen Few. *Information Dashboard Design*. O'Reilly, 2006.

[44] Andreas Fink and Stefan Voß. HotFrame: A Heuristic Optimization Framework. In *Optimization Software Class Libraries*, pages 81–154. Kluwer Academic Publishers, 2002.

[45] 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.

[46] José. E. Gallardo, Carlos Cotta, and Antonio J. Fernández. A Memetic Algorithm for the Low Autocorrelation Binary Sequence Problem. In *Genetic and Evolutionary Computation Conference*, pages 1226–1233. ACM, 2007.

[47] Michael R. Garey and David S. Johnson. *Computer and Intractability: A Guide to the Theory of NP-Completeness*. William H. Freeman, 1979.

[48] GATECH. The Traveling Salesman Problem.
http://www.tsp.gatech.edu.

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

[50] Fred Glover. Future Paths for Integer Programming and Links to Artificial Intelligence. *Computers and Operations Research*, 13:533–549, 1986.

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

[52] Fred Glover and Manuel Laguna. *Tabu Search*. Kluwer Academic Publishers, 1997.

[53] Fred Glover, Manuel Laguna, and Rafael Marti. Principles of Tabu Search. In *Approximation Algorithms and Metaheuristics*. Chapman & Hall/CRC, 2005.

[54] GraphViz. Graph Visualization Software.
http://www.graphviz.org.

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

[56] 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.

[57] 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.

[58] Steven Halim, Roland Hock Chuan Yap, and Felix Halim. Engineering Stochastic Local Search for the Low Autocorrelation Binary Sequence Problem. In *Principles and Practice of Constraint Programming*, pages 640–645, 2008.

[59] 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.

[60] 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.

[61] 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.

[62] Pierre Hansen. The Steepest Ascent Mildest Descent Heuristic for Combinatorial Programming. In *Congress on Numerical Methods in Combinatorial Optimization*, 1986.

[63] Pierre Hansen and Nenad Mladenovic. A Tutorial on Variable Neighborhood Search. *TR G-2003-46, GERAD*, 2003.

[64] Robert Harder. OpenTS.
http://opents.iharder.net, 2001.

[65] John Henry Holland. *Adaptation in Natural and Artificial Ecosystems*. MIT Press, second edition, 1992.

[66] John N. Hooker. Testing Heuristics: We Have It All Wrong. *Journal of Heuristics*, 1:33–42, 1995.

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

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

[69] Frank Hutter, Youssef Hamadi, Holger H. Hoos, and Kevin Leyton-Brown. Performance Prediction and Automated Tuning of Randomized and Parametic Algorithms. In *Principles and Practice of Constraint Programming*, pages 213–228, 2006.

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

[71] Toshihide Ibaraki, Koji Nonobe, and Mutsunori Yagiura. *Metaheuristics: Progress as Real Problem Solver*, volume 32 of *Operations Research/Computer Science Interfaces*. Springer, Berlin Heidelberg, New York, 2005.

[72] ILOG. ILOG.
http://www.ilog.com.

[73] David S. Johnson and Lyle A. McGeoch. The Traveling Salesman Problem: A Case Study in Local Optimization. In *Local Search in Combinatorial Optimization*, pages 215–310. John Wiley and Sons, 1997.

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

[75] 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.

[76] 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.

[77] Tomihisa Kamada and Satoru Kawai. An Algorithm for Drawing General Undirected Graphs. *Information Processing Letters*, 31(1):7–15, 1989.

[78] Scott Kirkpatrick, D. Gelatt Jr, and Mario P. Vecchi. Optimization by Simulated Annealing. *Science*, 220:671–680, 1983.

[79] 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.

[80] Gunnar W. Klau, Neal Lesh, Joe Marks, and Michael Mitzenmacher. Human-guided search. *Journal of Heuristics*, 2009.

[81] 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.

[82] Hoong Chuin Lau, Kien Ming Ng, and Xiaotao Wu. Transport Logistics Planning with Service-Level Constraints. In *19th National Conference on Artificial Intelligence*, pages 519–524, 2004.

[83] 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.

[84] Hoong Chuin Lau, Wee Chong Wan, Steven Halim, and Kaiyang Toh. A Software Framework for Fast Prototyping of Meta-heuristics Hybridization. *International Transactions in Operational Research*, 14(2):123–141, March 2007.

[85] Hoong Chuin Lau, Wee Chong Wan, Min Kwang Lim, and Steven Halim. A Development Framework for Rapid Meta-Heuristics Hybridization. In *International Computer Software and Applications Conference*, pages 362–367, 2004.

[86] Franco Mascia and Mauro Brunato. Techniques and Tools for Local Search Landscape Visualization and Analysis. In *Engineering Stochastic Local Search Algorithms*, 2009.

[87] Stephan Mertens. Exhaustive Search for Low-Autocorrelation Binary Sequences. *Journal of Physics A: Mathematical and General*, 29:473–481, 1996.

[88] Mertens, Stephan and Bauke, Heiko. Ground states of the Bernasconi model with open boundary conditions.
http://www-e.uni-magdeburg.de/mertens/research/labs/open.dat.

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

[90] Metaneos. Nug30 is Solved!
http://www.mcs.anl.gov/metaneos/nug30/nug30.pdf".

[91] Laurent Michel and Pascal van Hentenryck. Localizer++: An Open Library for Local Search. Technical Report CS-01-03, Department of Computer Science, Brown University, 2001.

[92] 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.

[93] Burkhard Militzer, Michele Zamparelli, and Dieter Beule. Evolutionary Search for Low Autocorrelated Binary Sequences. *IEEE Transactions on Evolutionary Computation*, 2(1):34–39, April 1998.

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

[95] Alfonsas Misevicius. Ruin and Recreate Principle Based Approach for the Quadratic Assignment Problem. In *Genetic and Evolutionary Computation Conference*, pages 598–609, 2003.

[96] Alfonsas Misevicius. Using Iterated Tabu Search for the Traveling Salesman Problem. *Informacines Technologijos Ir Valdymas*, 2004.

[97] Alfonsas Misevicius, J. Smolinskas, and A. Tomkevicius. Iterated Tabu Search for the Traveling Salesman Problem: New Results. *Information Technology and Control*, 34(4):327–337, 2005.

[98] Debasis Mitra, Fabio Romeo, and Alberto Sangiovanni-Vincentelli. Convergence and Finite-Time Behavior of Simulated Annealing. In *Conference on Decision and Control*, pages 761–767, 1985.

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

[100] Dagmar Monett-Diaz. *Agent-Based Configuration of (Metaheuristic) Algorithms*. PhD thesis, Humboldt University of Berlin, 2005.

[101] Greg Mori and Jitendra Malik. Recognizing Objects in Adversarial Clutter: Breaking a Visual CAPTCHA. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2003.

[102] NIST. e-Handbook of Statistical Methods.
http://www.itl.nist.gov/div898/handbook.

[103] Andreas Nolte and Rainer Schrader. A Note on the Finite Time Behaviour of Simulated Annealing. In U. Zimmermann, Ulrich Derigs, W. Gaul, Rolf H. Möhring, and K.P. Schuster, editors, *Operations Research Proceedings*, pages 175–180. Springer, 1996.

[104] Mika Nyström. Solving Certain Large Instances of the Quadratic Assignment Problem: Steinberg's Examples. Technical Report CSTR:2001.010, Caltech, 1999.

[105] Ibrahim H. Osman. An Introduction of Meta-Heuristics. In *Lawrence and Wilsdon (eds). Operational Research Tutorial Papers*, pages 99–122. Stockton Press, Hampshire, Publication of the Operation Research Society, UK, 1995.

[106] Ibrahim H. Osman. Metaheuristics: Models, Design and Analysis. In *5th Asia Pacific Industrial Engineering and Management System Conference, Kozan, E. (eds), Gold Coast, Australia, Dec 12-15th*, volume 1(2), pages 1–6, 2004.

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

[108] Joao Pedro Pedroso. Control of Search Parameters in Evolutionary Algorithms. In *International Symposium on Nonlinear Theory and its Applications*, pages 1265–1268, 1997.

[109] Paola Pellegrini, Daniela Favaretto, and Elena Moretti. On MAXMIN Ant Systems Parameters. In Marco Dorigo *et al.*, editor, *ANTS*, pages 203–214. Springer, 2006.

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

[111] Hartmut Pohlheim. Visualization of Evolutionary Algorithms - Set of Standard Techniques and Multidimensional Visualization. In *Banzhaf, W. (eds.). Genetic and Evolutionary Computation Conference*, pages 533–540, 1999.

[112] Steven David Prestwich. A Hybrid Local Search Algorithm for Low-Autocorrelation Binary Sequences. Technical report, Department of Computer Science, National University of Ireland at Cork, 2001.

[113] Steven David Prestwich. Exploiting Relaxation in Local Seach for LABS. *Annals of Operations Research*, 156(1):129–141, 2007.

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

[115] Ronald L. Rardin and Reha Uzsoy. Experimental Evaluation of Heuristic Optimization Algorithms: A Tutorial. *Journal of Heuristics*, 7(3):261–304, 2001.

[116] Victor J. Rayward-Smith, Ibrahim H. Osman, Colin R. Reeves, and George D. Smith. *Modern Heuristic Search Methods*. John Wiley and Sons, 1996.

[117] César Rego and Bahram Alidaee. *Tabu Search and Scatter Search*. Kluwer Academic Publishers, 2005.

[118] Christian M. Reidys and Peter F. Stadler. Combinatorial Landscapes. *SIAM Review*, 44(1):3–54, 2002.

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

[120] Mauricio G.C. Resende and Jorge Pinho de Sousa. *Metaheuristics: Computer Decision-Making*, volume 86 of *Applied Optimization*. Kluwer Academic Publishers, 2003.

[121] Celso C. Ribeiro and Pierre Hansen. *Essays and Surveys in Metaheuristics*, volume 15 of *Operations Research/Computer Science Interfaces*. Kluwer Academic Publishers, 2001.

[122] Mark F. Rogers, Adele Howe, and Darrell Whitley. Looking for Shortcuts: Infeasible Search Analysis for Oversubscribed Scheduling Problems. In *International Conference on Automated Planning and Scheduling*, pages 314–323, 2006.

[123] Simon Ronald. Distance Functions for Order-Based Encodings. In *IEEE International Conference on Evolutionary Computation*, pages 43–48, 1997.

[124] Simon Ronald. More Distance Functions for Order-Based Encodings. In *IEEE International Conference on Evolutionary Computation*, pages 558–563, 1998.

[125] H. H. Rosenbrock. An automatic method for finding the greatest or least value of a function. *The Computer Journal*, 3:175184, 1960.

[126] Johannes J. Schneider and Scott Kirkpatrick. *Stochastic Optimization*. Springer, 2006.

[127] 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.

[128] Marc Sevaux and Kenneth Sörensen. Permutation Distance for Memetic Algorithm. In *6th Metaheuristics International Conference*, 2005.

[129] Thomas Stützle. *Local Search Algorithms for Combinatorial Problems — Analysis, Algorithms, an New Applications*. PhD thesis, Technical University Darmstadt, Germany, 1999.

[130] Thomas Stützle, Mauro Birattari, and Holger. H. Hoos. *LNCS 5752: Engineering Stochastic Local Search*. Springer, 2009.

[131] Thomas Stützle and Marco Dorigo. ACO Algorithms for the Quadratic Assignment Problem. In *New Ideas in Optimization*, pages 33–50. McGraw-Hill, 1999.

[132] Thomas Stützle and Marco Dorigo. ACO Algorithms for the Traveling Salesman Problem. In *Kaisa Miettinen, Marko M. Mäkelä, Pekka Neittaanmäki, and Jacques Périaux. (eds). Evolutionary Algorithms in Engineering and Computer Science*, pages 163–183. John Wiley and Sons, Chichester, UK, 1999.

[133] 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.

[134] Thomas Stützle, Holger. H. Hoos, and Mauro Birattari. *LNCS 4638: Engineering Stochastic Local Search*. Springer, 2007.

[135] David Stynes. *Value Ordering for Offline and Realtime-Online Solving of Quantified Constraint Satisfaction Problems*. PhD thesis, National University of Ireland, 2009.

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

[137] Éric D. Taillard. Robust Tabu Search for Quadratic Assignment Problem. *Parallel Computing*, 17:443–455, 1991.

[138] Éric D. Taillard. Comparison of Iterative Searches for the Quadratic Assignment Problem. *Location Science*, 3:87–105, 1995.

[139] Éric D. Taillard and Luca Maria Gambardella. Adaptive Memories for the Quadratic Assignment Problem. Technical Report IDSIA-87-97, IDSIA, 1997.

[140] Éric D. Taillard and Luca Maria Gambardella. An Ant Approach for Structured Quadratic Assignment Problem. Technical Report IDSIA-22-97, IDSIA, 1997.

[141] Lingjia Tang and Paul F. Reynolds Jr. User Guide for Chained Use of Optimization Techniques to Gain Insights. 2007.

[142] Reinaldo A.L. Trujillo. ACOVis: Bsqueda Guiada por el Usuario mediante la Visualizacin en Optimizacin por Colonias de Hormigas. 2009.

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

[144] Edward Tufte. *The Visual Display of Quantitative Information.* Graphic Press, 1983.

[145] Edward Tufte. *Envisioning Information.* Graphic Press, 1990.

[146] Edward Tufte. *Visual Explanations: Images and Quantities, Evidence and Narrative.* Graphic Press, 1997.

[147] Edward Tufte. *Beautiful Evidence.* Graphic Press, 2006.

[148] Abhisek Ukil. Low autocorrelation binary sequences: Number theory-based analysis for minimum energy level, Barker codes. In *Digital Signal Processing*, volume 20, pages 483–495, 2009.

[149] Pascal van Hentenryck and Laurent Michel. *Constraint-Based Local Search.* MIT Press, Cambridge, London, 2005.

[150] Stefan Voß. Meta-Heuristics - The State of the Art. In *Nareyek, Alexander (eds). Local Search for Planning and Scheduling*, volume LNAI 2148, pages 1–23. Springer-Verlag, London, UK, 2001.

[151] Stefan Voß, Silvano Martello, Ibrahim H. Osman, and Catherine Roucairol. *Meta-Heuristics - Advances and Trends in Local Search Paradigms for Optimization.* Kluwer Academic Publishers, 1998.

[152] Colin Ware. *Information Visualization: Perception for Design*. Morgan Kaufmann, second edition, 2004.

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

[154] Wayne L. Winston. *Operations Research: Applications and Algorithms*. Thomson Brooks/Cole, 2004.

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

[156] 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.

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

[158] Zhang Zhongzhen. A New Method for Quickly Solving Quadratic Assignment Problems. Technical report, School of Management, Wuhan University of Technology, 2009.

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

Appendix A

COP Details

In this appendix, we elaborate the details and compare (see Table A.1) the COPs discussed in this thesis. The problem definitions for these COPs are already shown in Chapter 7.

Traveling Salesman Problem (TSP)

Applications

Most of the works on TSP are not motivated by direct applications, but rather because of its simple problem definition, its popularity, and its hardness serve as an ideal platform for studying and benchmarking algorithms.

Applications in transportation are the most natural setting for TSP, e.g. finding a shortest possible trip for a salesman starting from his home city, going through a given set of customer cities once, and then return to his home city.

TSP has other interesting applications, e.g. scheduling a route for a machine to drill holes in a circuit board. The holes are the 'cities' and the objective is to reduce manufacturing costs by minimizing the travel time to move the drill head from one hole to the next. If the surface of the circuit board is sloped/tilted, then the travel costs for going up or down are different and thus it is an instance of Asymmetric TSP.

The Progress of Exact Algorithms

Techniques like Branch & Cut or Cutting Plane are the leaders for solving TSP.

As of 13 July 2009, the largest TSP instance that has been optimally solved is pla85900 in 2006 [48] with 85900 cities.

The Progress of Non-Exact Algorithms

Although many TSP instances can be optimally solved by exact algorithms within 'hours' or 'days', non-exact algorithms for TSP are still required. The objectives for these non-exact algorithms are (1). to reach *near* optimal solutions in a *much shorter time* than exact methods for small to medium TSP instances, (2). to obtain reasonable quality solutions for large TSP instances (e.g. 1904711 cities world-tour [48]) where

exact algorithms are still infeasible, and (3). as benchmark for studying the non-exact algorithms performance.

Some well-known heuristics for TSP are greedy nearest neighbor heuristic, k-Opt edge swap heuristic, Lin-Kernighan heuristic, etc. These heuristics have been embedded inside many other TSP SLS solvers to further improve the search quality, e.g. 2/3-Opt edge swap heuristic inside ILS [133].

In early 1990s, TSP was used as the initial test study for ACO algorithm due to the natural mapping between ants foraging behavior with traveling salesman behavior. Both involves routing and are looking for shortest path [36, 132].

Natural Representations and Typical Local Moves

The most natural representation is perhaps the permutation of cities that maintains the all-different constraint, e.g. 1-5-3-4-2 represents a tour of 5 cities starting from city 1, going to 5, 3, 4, 2, and then cycling back to city 1.

There are two common local moves for modifying TSP solutions that maintain the all-different constraint. First is the $O(n^k)$ Swap k-Vertices, usually $k = 2$. This move is not good as it causes many tour crossings and will most likely degrade solution quality rather than improving it. Alternative: $O(cn)$ Swap k-Edges, usually $k = 2$ or $k = 3$. This is a more natural local move. Its neighborhood size is smaller as we can restrict an edge e connected to a vertex v to be swapped only with other c-shortest edges connected to v – swapping e with a long edge will likely degrade tour quality [129].

Remarks

The origin of TSP is obscure, but it has been around for quite some time.

TSP has been extensively studied and researched. Thus, it is hard or even unlikely to find better TSP tours than the best published ones for well-known benchmark instances, e.g. TSPLIB [119, 143].

Other than classical TSP, researchers study TSP variants, e.g. TSP with Time Windows, Non Euclidean TSP, TSP with constraints (certain edges are forbidden), etc. See [73] for a more complete discussion about TSP.

Quadratic Assignment Problem (QAP)

Applications

The term QAP was first introduced in 1957 by Koopmans and Beckmann [140], when they derived it as a mathematical model of assigning a set of economic activities to a set of locations.

Other applications includes: Facility (Hospital) Layout, Berth Location, Typewriter Design, etc, where one wants to put facilities/berths/keys in such a way that minimizes the movement of people/goods/fingers. Usually, good QAP solution has facilities/berths/keys that have high flow/interaction placed close to each other.

The Progress of Exact Algorithms

As of 13 July 2009, exact algorithms can only solve QAP instances up to $n \leq 40$, e.g. tho40 in 2009 [158], ste36 in 1999 [104], or nug30 in 2000 [90]. To find the optimal solution for nug30, the Branch & Bound computation required 11892208412 nodes, using on average 650 parallel workstations in 7 full days to verify that 6124 is the optimal value for the instance, and found the optimal permutation which is exactly the permutation that has been found by an SLS algorithm: Ro-TS many years back in 1990 [137].

Among 135 QAP instances in QAPLIB [22, 114], most instances with $n \leq 40$ have been proven to be optimal using exact algorithms, whereas the BK OVs for the rest ($40 < n \leq 150$) are obtained using various SLS algorithms. This fact and the knowledge that the largest instance solved is of size 40 (compared with 85900 for TSP) make QAP one of the most difficult COPs.

The Progress of Non-Exact Algorithms

The inherent complexity of QAP has attracted several algorithm designers to test their SLS implementation on QAP. Since early 1990s, there are variants of SLS algorithms developed to solve QAP, especially variants of Tabu Search (TS).

The original Strict TS (S-TS) idea introduced by Glover in 1980s [52] was used by Skorin-Kapov to attack QAP [137]. Taillard then extended it into the Robust TS (Ro-TS) [137]. Battiti & Tecchiolli continued this trend by introducing a concept of Reactive TS (Re-TS) [15]. Ahuja *et al.* [5] proposed a Very Large Scale Neighborhood (VLSN) approach, in which they experimentally show that TS with k-Opt (large) neighborhood is better than with 2-Opt (considered small) neighborhood. This approach is good but the computation time needed is tremendously big. Recently, Misevicius proposed another search strategy called Ruin & Recreate (R&R) [95]. His TS algorithm using this search strategy was able to improve several best known solutions for QAP instances; especially taixxc grey instances and taixxb (real life like instances).

Other approaches for solving QAP using ACO or other SLS algorithms were proposed in [138, 139, 140, 131], etc. However, BK solutions for $n > 40$ currently reported in QAPLIB are usually found by the early versions of TS for QAP variants, e.g. Ro-TS, Re-TS, or TS with R&R strategy.

Natural Representations and Typical Local Moves

QAP solution is best represented as an array of assignment. An array index represents a facility and its value represents the location assigned to this facility.

To automatically maintain the permutation (all-different) constraint, the most appropriate local move is the $O(n^k)$ Swap k-locations. Typically, $k = 2$ or $k = 3$.

Remarks

As with TSP, this COP: QAP is also well researched too.

Low Autocorrelation Binary Sequence (LABS)

Applications

The LABS problem was first posed in the Physics community in 1960s. It has applications in many communication and electrical engineering problems, including high-precision interplanetary radar measurement [87].

The Progress of Exact Algorithms

The LABS problem is a challenging problem for exact algorithms. It troubles constraint programming techniques due to its symmetrical nature and limited propagations.

As of 13 July 2009, LABS problem can only be solved optimally up to $n = 60$ [30] using the Branch & Bound (B&B) algorithm by Mertens [87] in 2005.

The Progress of Non-Exact Algorithms

The LABS problem is said to pose significant challenge to local search methods. It is said that stochastic search procedures are not well suited to find these 'golf holes'-like global optima [87]. This statement is now no longer true with recent publications.

In 2001, Prestwich [112], proposed a hybrid B&B and local search, called Constrained Local Search (CLS). CLS is faster than Mertens's B&B approach [87] in finding optimal LABS solutions for $3 \leq n \leq 48$.

In 2006, Dotú and van Hentenryck [38] proposed a *simple* SLS algorithm: Tabu Search (TS) *with frequent restarts*. This TS could find optimal LABS solutions for $3 \leq n \leq 48$ much quicker than Merten's B&B [87] or Prestwich's CLS [112]. It was roughly on par with another good SLS solver for LABS problem: Kernighan-Lin [21] (2003).

In 2007, Gallardo *et al.* [46] proposed an SLS: MA_{TS}, combining a Memetic Algorithm with similar TS as in [38]. MA_{TS} was shown to be "one order of magnitude" faster than the *pure* TS [38] and was the fastest LABS solver in 2007.

In 2008, we have shown how IWBBA (see Chapter 6) using an SLS engineering tool Viz (see Chapter 5 and 6) can be used to successfully engineer a new state-of-the-art SLS algorithm for LABS problem starting from the TS algorithm proposed in [38]. This result is reported in Chapter 7 and in [58].

Natural Representations and Typical Local Moves

Solutions for LABS problem can be represented as a bit string, with simple adjustment in the objective function to regard 0/1 in bit string as -1/+1 in LABS solution.

The natural local move for locally modifying the bit string is the $O(kn)$ k-bit flip neighborhood. Usually $k = 1$.

Remarks

We managed to obtain the state-of-the-art SLS algorithm for LABS: TSv7 [58].

Item	TSP	QAP	LABS
Natural Representation	permutation/tour	permutation/assignment	bit string
Unique Search Space Size	$n!/(2n)$	$n!$	2^n - symmetries
Constraints	all-different	all-different	no constraint
	- each city is visited once	- one-to-one assignment	-
Absolute Computation of $g(n)$	$O(n)$	$O(n^2)$	$O(n^2)$
Incremental Computation of $g(n)$	$O(1)$	$O(1)$	$O(n)$
Neighborhood N (good)	Swap k-Edges $O(cn)$	Swap k-Locations $O(n^k)$	Flip k-Bits $O(kn)$
Neighborhood N (alternative)	Swap k-Vertices $O(n^k)$	-	-
Reasonable size[a] of n	200 cities (baseline)	$\approx$ 100 facilities/locations	$\approx$ 50 bits
FDC r_{FDC}	straightforward (high r_{FDC})	difficult (low r_{FDC})	difficult (low r_{FDC})
Distance function $d(s_1, s_2)$	bond	Hamming	Hamming
Local Optima Distribution	Big Valley	Spread	Spread but bounded
Local Optima Variability	Low inside Big Valley	Type A: smooth	Very high generally
	High outside Big Valley	Type B: rugged	-
Known Issues	easy to get good results	large neighborhood space	discrete $g(n)$
	hard to get optimal result	-	irregular fitness landscape
Benchmark Library	TSPLIB [119, 143]	QAPLIB [22, 114]	CSPLIB #005 [30]
Important References	[48, 73, 45, 132, 133]	[137, 15, 138, 57, 61]	[87, 21, 38, 46, 58]

Table A.1: The Comparison of Various Combinatorial Optimization Problems used in this thesis

[a]This n is roughly measured using the following methodology: We set kraA200 of TSPLIB as benchmark. Our ILS_T implementation solves kraA200 on average 23 seconds (40000 iterations) in our computer (2.33 GHz Core2 Quad PC). We then find the largest QAP and LABS instance size on which our best SLS implementations do not exceed 23 seconds on average.

Appendix B

SLS Details

In this appendix, we list down popular **Parameters**, Components, and SearchStrategies of the SLS algorithms used in this thesis (TS and ILS).

Some abbreviations mentioned here are already explained in *List of Abbreviations* in front of this thesis. The rest are elaborated here.

Tabu Search (TS)

Short Background

Tabu Search (TS) is a trajectory-based SLS proposed by Glover in early 1980s [50, 52, 51, 53]. Since then, TS has been widely used to attack many COPs. The strength of TS lies in its capability to escape local optimality – TS alters the neighborhood using tabu mechanism so that TS is discouraged from re-visiting explored solutions.

Basic Algorithm

```
Tabu-Search()
1  CurS ← BF ← InitS
2  while TermCrit are-not-satisfied
3  do BestMove ← Best(N, TabuM(TT), AspC, CurS)
4     CurS ← BestMove(CurS)
5     TabuM.SetTabu(CurS, BestMove, TT)
6     if Better(CurS, BF)
7       then BF ← CurS
8     SearchStrategies
9  return BF
```

Explanation of basic TS: Starting from an initial solution, pick the best local move to the best neighbor which is either (1) not tabu or (2) tabu but aspired. The tabu mechanism will then discourage re-visitation of this solution for the duration of tabu tenure. This is to prevent cycling and forces the search to explore other regions. This process is repeated until termination criteria are met.

Configuration

Parameter Values

Tabu Tenure (TT). **TT** is one of the most influential parameter in TS. Setting **TT** as a **static** value throughout the set of instances is usually a bad idea since different instance size n typically requires different **TT** value. Setting **TT** as a **function of instance size** n is much better, e.g. Ro-TS for QAP sets **TT** to be within $[90 \dots 110]\% * n$ throughout the search [137].

Components

Neighborhood N. The size and type of N is problem specific, e.g. $2/3/k$**-opt swap moves**, k**-bit flip moves**, etc. The larger and more complex N is usually better as shown in **VLSN (Very Large Scale Neighborhood)** [5]. However larger N is slower – a tradeoff between running time and quality.

Tabu Mechanism TabuM. TabuM is also called as Reverse Elimination Method (REM). To completely prevent solution cycling during the duration of **TT**, **tabu solution** is the best TabuM implementation. But it is usually slow and inefficient. It is much more practical to **tabu recently applied moves** or **tabu the attributes of recent solutions**. TabuM can be implemented using a 'linked list' or a 'circular array' but it perhaps best implemented using a 'hash table'.

Aspiration Criteria AspC. In TS, the role of AspC is not as significant as TabuM (AspC is optional). However, AspC may help improving the overall search quality. Thus, if we seek a very good result, this component should be adjusted properly. Usually AspC is in form of **best ever criterion**, where tabu moves leading to best ever solution are allowed. Some other form of AspC: **diversification based on frequency/history** to allow tabu moves to be considered when those tabu moves lead to solutions that are rarely visited for diversification purpose.

Termination Criteria TermCrit. TermCrit is a component that can affect the performance too. When and how we terminate a TS run affect its apparent performance. Usual termination criteria are: **maximum time**, **maximum iteration**, or **target objective value**.

Initial Solution InitS. A good initial solution InitS created by **problem-specific construction heuristic** may help TS to reach good region quickly, but it may create a tendency of premature convergence. Sometimes, a **random construction heuristic** is used instead. However, the effect of InitS is not too significant in long run as by that time TS will have explored regions far from InitS anyway.

Search Strategies

Robust Tabu Search Ro-TS [137, 138]: TS randomly change **TT** within a range $[low \dots high]$ for every predetermined periods. This is to enhance TS capability in escaping local optima.

Reactive Tabu Search Re-TS [15]: TS adaptively adjust **TT** length: lengthen **TT** if it is experiencing solution cycling and shorten **TT** if it is not improving. Perhaps better than Ro-TS when **TT** adaptation is done properly.

Path Relinking [52]: TS uses good local optima as the guiding force to create new paths, hopefully while TS is traversing along this path, it hits better results.

Oscillation Strategies [52]: TS alternates between feasible and infeasible region because the optimal usually lies within the boundary of these two extremes.

Ruin and Recreate RR [95]: TS performs strong diversifications where $x\%$ of the structure in the current best solution are retained while the remaining structures are randomly perturbed. Then, TS resumes its search. This strategy is shown to be working well for real-life and real-life-like QAP instances.

Some Applications

1. Traveling Salesman Problem [73, 79, 96, 97, 60].
2. Quadratic Assignment Problem [137, 138, 15, 5, 95, 57, 61].
3. Low Autocorrelation Binary Sequence [38, 58].

Iterated Local Search (ILS)

Short Background

Iterated Local Search (ILS) is an SLS algorithm that combines the power of simple local search plus controlled diversification/intensification in form of Perturbation (Ptb)/AcceptanceCriteria (AccC) mechanism. For more details, refer to [51, 133].

Basic Algorithm

```
Iterated-Local-Search()
 1  CurS ← BF ← LS(InitS)
 2  while TermCrit are-not-satisfied
 3  do TempS ← Ptb(Ptb-Str)(CurS)
 4     TempS ← LS(TempS)
 5     CurS ← AccC(TempS, CurS)
 6     if Better(CurS, BF)
 7        then BF ← CurS
 8     SearchStrategies
 9  return BF
```

Explanation of basic ILS: Given an initial solution, locally optimize it to reach the first local optimum. Now, start the ILS loop: perturb the local optimum according to the perturbation strength, locally re-optimize the perturbed solution again, hoping that the re-optimized solution arrives at a better (or more promising) solution. Then, use acceptance criteria to decide whether the newly found local optimum is accepted or stick with the old one. This ILS loop is repeated until termination criteria are met.

Configuration

Parameter Values

Perturbation Strength Ptb-Str. **Ptb-Str** determines how radical a solution is perturbed. This parameter is either a **constant** or a **function of instance size**.

Components

Local Search LS. This is the heart of the ILS algorithm. The choice of LS is problem specific. It can be a simple gradient descent heuristic or even another SLS algorithm.

Perturbation Mechanism Ptb. Ptb(**Ptb-Str**)] must yield solutions that are not easily reversed by the LS, otherwise a severe solution cycling issue may arise, e.g. **double bridge move for TSP**, **ruin and recreate**, **random restart**, etc.

Acceptance Criteria AccC. AccC determines whether the effort of Ptb(**Ptb-Str**) and LS pair should be accepted or discarded. Using **better only** AccC, ILS only accepts new local optimum if it is better than the previous local optimum (before perturbation and local search phase), otherwise the next perturbation will perturb the old local optimum, making the search more focused on the good regions only. Using **random walk** AccC, ILS always move to newly found local optimum, this can be good or bad depending on the fitness landscape characteristics. Using **small probability** AccC, ILS is allowed to move to worse local optimum with small probability.

The options for **Initial Solution InitS** and **Terminating Condition TermCrit** in ILS are similar like in TS.

Search Strategies

FDD-Diversification [133]: ILS performs *stronger* diversification when it seems stuck in a deep local optimum and the current Ptb(**Ptb-Str**) and AccC pair is not strong enough. This strategy is shown to work for the Big Valley region in TSP.

Some Applications

1. Traveling Salesman Problem [133, 59, 61]
2. Quadratic Assignment Problem [129]

Remarks

For the other SLS algorithms that are not discussed in this thesis, e.g. Ants Colony Optimization (ACO), Simulated Annealing (SA), Genetic Algorithm (GA), see the references like 'Handbook of Metaheuristics' [51].

Appendix C

Human Strengths

Despite the increasing demand to transfer our (human) works to computers in order to simplify our life, there are still a lot of human tasks that cannot be (or still poorly) done by *current*[1] computer, such as visual perception and intelligence.

To illustrate the strength of human over computer, we highlight the recent research in CAPTCHA [4] (Completely Automated Public Turing Test to Tell Computers and Humans Apart)[2]. CAPTCHA utilizes an idea that: "It is easier for computer to generate visualization than to derive information from the generated images".

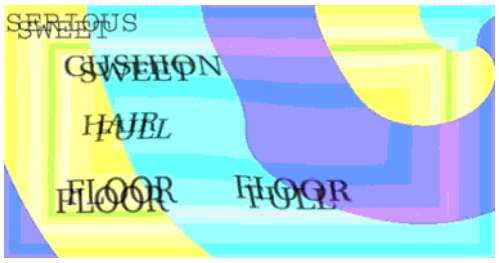


Figure C.1: Gimpy [4]: What are the words written here?

While it is considered easy for human to read the distorted and corrupted words in Figure C.1[3], it is difficult (but not impossible, see [101]) for the current state-of-the-art Optical Character Recognition (OCR) algorithms to correctly decipher the words.

This CAPTCHA is called 'gimpy'. Gimpy *randomly* grabs few letters or numbers and then distorts those using different colors, stretching the letters, adding extra noises, etc. Despite such nasty alterations, most human pass this test quite easily[4].

Another case of the superiority of human visual perception and intelligence in deriving information is shown in another CAPTCHA called 'pix' (See Figure C.2). Pix grabs four pictures with the same label (these pictures are already labeled by another

[1]No one knows whether future technologies will be able to take over the areas where human is currently better. When that happens, Human Computer Interaction must be redefined.

[2]Nowadays, many web services use CAPTCHA to verify that the user is really human instead of a malicious computer program. For example, when a user sign up for a free e-mail account, he will be asked to do what human is known to be good at but difficult for machine. This is to prevent the free e-mail account to be auto registered/spammed by malicious web-bots spammers.

[3]Answer: Cushion, Floor, Full, Hair, Serious, Sweet.

[4]It is true that some GIMPY are quite hard that even normal human has difficulties. The researches to create better CAPTCHA as well as better OCR algorithms are still ongoing.

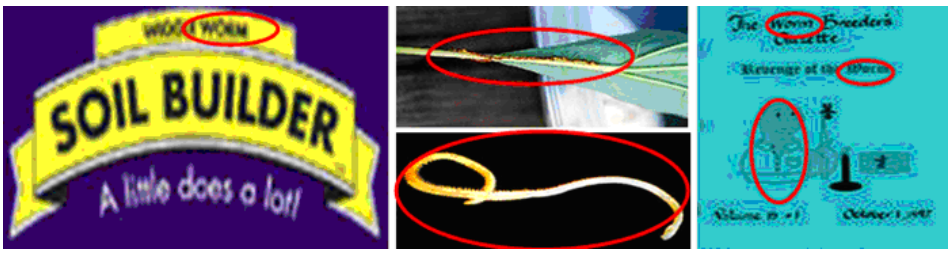


Figure C.2: Pix [4]: What is the common object in these 4 sub-figures?

human beforehand) and ask the user to find a single word that best describes the main object of the four pictures.

Human can easily answer: 'worm' (circled), but at the moment, to the best of our knowledge, there is yet a computer algorithm that can successfully connect the correlation between those distinct pictures.

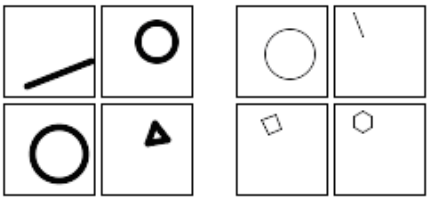


Figure C.3: Bongo [4]: What is the major difference between these two figures?

In Figure C.3, another CAPTCHA called 'Bongo' is shown. In this 'IQ test', the users are asked to tell the major difference of the 4 pictures on the left side versus the 4 pictures on the right. The answer is easy for human: Pictures on the left are drawn with thick lines whereas the pictures on the right are drawn with thin lines. It seems hard to create a dedicated algorithm to accomplish the same thing.

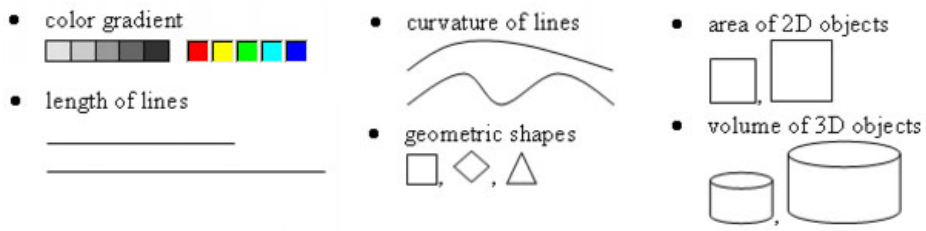


Figure C.4: Examples of visual features that are easily identified by human.

Yet another case[5] is shown in Figure C.4. Human can easily distinguish several visual features of whether a specific object in the given picture has a rectangle or triangle shape, curvy or straight, big or small, and so on. Computer needs a sophisticated algorithm to achieve the same feat and currently still not perfect.

So, although computers are much faster than human in numerical computations, human are still far better at carrying out low-level tasks such as speech and image recognition (shown above). This is due in part to the massive parallelism employed by human brain, which makes it easier to solve such problems.

[5]There are others, e.g. peekaboom, espgame, etc.

Index