Contents

Chapter 2

Background

If I have seen a little further, it is by standing on the shoulders of Giants.[1]
— **Isaac Newton**

This chapter provides background materials for this thesis. We briefly introduce the $\mathcal{NP}$-hard Combinatorial (Optimization) Problems (COPs), trade-offs between exact and non-exact algorithms, and then extensively discuss various aspects of non-exact Stochastic Local Search (SLS) algorithms. The materials presented in this chapter is necessary to appreciate the rest of this PhD thesis, especially the parts about terms and notations.

2.1 $\mathcal{NP}$-hard Combinatorial (Optimization) Problems

> Grace is the owner of a small food delivery service in Singapore. Everyday, orders come via online booking and she needs to plan a schedule to serve all these orders using her only car. With approximately 30 customers per day scattered around the neighborhood, Grace faces about 30! (a very big number) possible routes to choose from. Each route requires different amount of time and operating cost (fuel). In order to maximize profit, Grace must choose the shortest route which will save time and fuel. This computational problem can be modeled using a Traveling Salesman Problem (TSP), where the objective is to minimize the sum of distance traveled by the car. By using a good solver for this problem, Grace can save operating cost and thus increase his company's profit.

The simplified[2] example above is a single-objective Combinatorial (Optimization) Problem (COP) since the solutions for the TSP problem above are combinatoric[3] — a set of car routes. COPs like this are found in many other real-life applications, e.g. application of Traveling Salesman Problem to a real-life circuit drilling problem [29, 48], application of Quadratic Assignment Problem to a real-life hospital layout problem, keyboard layout [47, 39], application of Job Shop Scheduling Problem (JSSP) to a real-life scheduling problem [18], and many other applications in the field of Operations Research (Management Science), Engineering, Science, Computational Biology, etc. A more complete reference of classical COPs are listed in the Appendix A.

Solving COPs can means business. There are companies that provide optimization services like ILOG [28] where they model their clients' real-life problems into COPs and optimize the solutions

[1] This is the saying of Isaac Newton. He said that academic research can only advance (we can see further) by learning what the previous researchers has done (the Giants). This chapter, hopefully, can be the strong foundation (the Giants) of this thesis from which we can see further.

[2] [3] make comments on creating good solvers for overly abstract or over simplified problems that do not reflect the true real life settings where the solver for such problem will be deployed. Thus, no matter how good the solver for these abstract problems, it is just "a good solution for the wrong or not important problems". In this thesis, we keep highlighting the applicability of the COP that we attack in the context of real-life settings.

[3] There are optimization problems where the solutions are continuous. This class of optimization problem is easier to solve as one can use established linear programming techniques like SIMPLEX [53]. There are also multi-objectives optimization problems where we need to optimize more than one (usually conflicting) objectives. This class of optimization problem is usually solved using population based SLS algorithms such as Evolutionary Algorithms. Continuous and multi-objectives optimization problems are outside the scope of this thesis

for these COPs. The results can be used by the clients to decide better business decisions which in turn boosts the clients' company performance. Depending on how large the project is, even small improvements can mean multi-million dollars revenues.

A formal definition of a *minimizing*[4] COP is as follows:

Find solution s that minimize the objective function $g(s)$ of a COP instance π
Such that: $s \in$ search space $S(\pi)$ and s satisfies all the hard constraints $\in C$

Problem Instance π : An instance of the COP, e.g. lin105.tsp, pcb442.tsp (TSPLIB [42]); tai40a.dat, tai80b.dat (QAPLIB [5]); C101.txt, R201.txt (Solomon VRPTW instances [45]), etc. $|\pi| = n$ denotes the size of instance π.

Solution s : $s(x_1, x_2, \ldots, x_n)$ is an n-dimensional combinatorial solution of π. Note that here s does not have to satisfy all the constraints C of the COP — the term 'solution' in this thesis refers to both feasible or infeasible solution. In the literature, solution defined like this is also called as 'instantiation'.

Search Space $S(\pi)$: The set of possible solution s of π. $|S(\pi)|$ denotes the size of search space $S(\pi)$.

Constraint C : The set of rules that determine whether a certain solution s is feasible or not. There are two distinctions: hard-constraints are those that cannot be violated at any cost whereas soft-constraints can be violated but every violation incurs a penalty which will decrease the quality of the solution. Hard-constraints causes some solutions to be 'infeasible'. However, sometimes we purposely violate soft-constraints in order to have good quality solutions even after small penalty.

Objective Function $g(\pi, s) : S \rightarrow \Re$: The objective value OV for a solution s of COP instance π is $g(\pi, s)$, but usually denoted as $g(s)$ only. This function is problem-specific. In the literature, objective function is also known as fitness function. In this thesis, we use the term objective function and we restrict it to a single-objective only.

Considering that COPs has many practical usage and can bring profit and efficiency, it is thus very important to have a good solver for these COPs. However, finding a good solver is still a real challenge, despite the impact of recent advances in computer technology.

The major source of the hardness of these COPs is because many of the COPs are classified as $\mathcal{NP}$-hard[5]. The complexity theory states that by using deterministic machines (like the one that we have now), there is no[6] polynomial time algorithm available to find the optimal solution(s) for these $\mathcal{NP}$-hard COPs[7]. In other words, no algorithm is expected to solve them exactly within reasonable time limit, since within a so called 'reasonable' time limit, we will only able to evaluate a relatively small fraction of all possible solutions of the given $\mathcal{NP}$-hard problem [16].

The facts that many of these important COPs are $\mathcal{NP}$-hard render exhaustive search approach (e.g. brute force try-all) infeasible as the problem instance size increases[8]. The hardness of these problems is the motivation for research in optimization algorithms and numerous proposals have been published as attempts to address this issue. We discuss the proposals below.

[4] Without loss of generality, every minimizing COP can be converted to maximizing COP by negating the objective value.

[5] There exist COPs that are not classified as $\mathcal{NP}$-hard, for example: Minimum Spanning Tree (MST) problem, where one needs to select $V - 1$ edges out of E edges such that there is no cycle and the total length of selected edges is minimum. For MST problem, there exist efficient greedy algorithms, such as: Kruskal or Prim Algorithms [7], which are adequate for solving this problem in polynomial time. Another example is shortest path problem for graph with positive weighted edges as there exists efficient polynomial time algorithm such as Dijkstra [7] to solve the problem. These kinds of non $\mathcal{NP}$-hard COPs are outside the scope of this thesis.

[6] There is an ongoing discussion about quantum or DNA computing that can process non-deterministic problems in polynomial time [**??**]. However, such theoretical machines are not available at the moment and not predicted to appear in the near future.

[7] Even for these $\mathcal{NP}$-hard COPs themselves, the 'hardness' varies. For example, QAP (as of 25 July 2007, exact algorithms can solve QAP of up to 30 facilities/locations [**??**]) is considered 'harder' to solve than TSP (as of 25 July 2007, exact algorithms can solve TSP of up to 15.000 cities [**??**]).

[8] A simple way to measure the maximum instance size n that still do-able with exhaustive search is to start with a small n, increase n by 1 step by step, and after one point, the run time required by the exhaustive search algorithm to solve n and $n + 1$ will be noticeably *very* different. This is the rough upper bound of n that can be solved in 'reasonable' time by this algorithm.

2.2 Algorithms for Attacking COPs

There are various proposed algorithms for solving COPs, but basically they can be classified into two extreme paradigms: *exact* versus *non-exact* algorithms.

2.2.1 Exact Algorithms

Exact algorithms are clever version of exhaustive search (brute force try-all) approach. These exact algorithms are complete in the sense that the existence of a feasible solution and then the optimal solution can be determined with certainty once such exact algorithm is successfully terminated.

Examples of exact algorithms are techniques[9] such as Branch and Bound (B&B), Branch and Cut (B&C) [53], Constraint Programming (CP) [**??**]. In B&B or B&C, we prune the search sub-tree once sufficient evidence is gathered which guarantee that there is no solution under those sub-tree which will give better result than the one currently known. With such pruning techniques, B&B and B&C effectively search on much smaller but promising (in the sense that the optimal solution is probably there) search space only. In CP, we perform domain reduction, constraint propagation, and systematic search to find best feasible solutions.

While exact algorithms are guaranteed to deliver the optimal solution (if any[10]), these exact algorithms have problems with their running time. Typically, to solve a COP of size n, these exact algorithms still requires either exponential time $O(k^n)$ for some constant k or factorial time $O(n!)$ even though these algorithms do not actually explore the whole search space. This huge time complexity renders these algorithms infeasible for reasonably big n, which, unfortunately, are commonly found in real-life settings.

There are quite a number of research to advance exact algorithms in terms of better speed, tighter bounds, better cutting planes, etc. However, notwithstanding its importance, it seems that until someone found a way to make $\mathcal{P} = \mathcal{NP}$, exact methods are not the best approach for solving *large* COP instances in *reasonable* time.

2.2.2 Non-Exact Algorithms

For practical usage, especially in real time setting, faster techniques are needed. This is where the other extremes, the non-exact algorithms, emerge, such as: Approximation Algorithms, Heuristics, and Stochastic Local Search/Metaheuristics. A very reasonable assumption that a user will already be satisfied if he can consistently obtain good enough[11] solutions within reasonable time, leads researchers to develop various non-exact algorithms. In this extreme, one sacrifices the guarantee of optimality (completeness) for (much) faster speed. That is, in this setting we are interested in good enough, soon enough, cheap enough solutions.

There are more drawbacks than just sacrificing the guarantee of optimality. By using non-exact algorithms, we may not be able to state how close a particular solution quality produced by the non-exact algorithm w.r.t the optimal solution. Non-exact algorithm does not even guarantee to find a feasible solution at all if the problem has very few feasible solution.

Approximation Algorithms

Approximation algorithms are simple, polynomial time algorithm that gives X-approximation of the solution for a problem, where X is the performance guarantee of that approximation algorithm. Typically, one needs to provide a rigorous proof to support such claim. X is always > 1 and researchers are trying to reduce the gap as close to 1 as possible ($X = 1$ for an exact algorithm). This kind of algorithm is successful for certain COPs because such COPs become easy if some constraints are relaxed. For example, the 2-approximation algorithm for Euclidian Traveling Salesman Problem can be derived from the polynomial time Minimum Spanning Tree algorithms [7]. By using this 2-approximation algorithm, user is guaranteed to obtain Euclidean TSP solution that will not be worse than two times the quality of the optimal solution[12].

[9] There are other techniques like Integer Programming (IP), Mixed Integer Linear Programming (MILP), Column Generation, etc. Interested readers can browse books like [53].

[10] We should consider rare cases where the given COP itself is very over-constrained such that there is no single feasible solution exists, thus no optimal solution at all.

[11] 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.

[12] The 2-approximation algorithm is actually still 'very bad'. Imagine if the optimal cost is 1 million $ and now the 2-approximation algorithm returns 2 million $. Will we use the given solution or try another approach and see

However, not all approximation algorithms are suitable for practical applications. For example, most people would not be impressed by a 10-approximation algorithm where the quality of the solution may end up 10 times worse than the quality of the optimal solution. Also, for some approximation algorithms, the polynomial run time can be quite bad, e.g. $O(n^7)$. For many other COPs, approximation algorithm is not possible due to the complexity of the COPs. For those reasons, other more-generic non-exact algorithms are required.

Heuristics and Stochastic Local Search/Metaheuristics

Heuristics are simple, usually greedy techniques which seek (and hopefully find) good solutions at a reasonable computational cost, e.g. in low order polynomial time. Heuristics is a more general form of approximation algorithm where this time there is *no* performance guarantee.

Heuristic algorithms are usually arise from simple intuitive reasoning on the problem at hand: think greedily and examine the characteristic of good solutions. For example: good TSP tours have short edges. Thus, starting from any city and then greedily picking the nearest neighbor one by one to complete the TSP tour intuitively will produce a reasonably good solution for TSP (except for the last few edges). This greedy nearest neighbor heuristic has no performance guarantee but it performs quite well on several TSP instances, typically better than the 2-approximation algorithm [??].

The major weakness of heuristics is that it is confined into small search space only, which may be severely sub-optimal, unless this heuristic is very tailored to solve the COP well[13]. A heuristic does not actually 'search' the COP search space, it just zoom in into a specific solution based on the heuristic 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) a.k.a metaheuristics come into the picture. For simplicity, in this thesis we will refer any form of algorithm that navigates along the search space (which will be referred as fitness landscape in the later sections) as SLS algorithm. We discuss the SLS algorithms in-depth in Chapter 2.3.

2.2.3 Comparison of the Two Extremes

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

The slowness of exact algorithms do not imply that they are useless. When optimality is a hard constraint or if the problem is so overly constrained that we want to find whether there exist a solution, we do not have any other choice other than using exact algorithms. Also, logically, exact algorithm should be the algorithm of choice when the problem size that we are dealing with is *small enough* for exact algorithms to be run within the given running time.

However, when the instance size of the COP being attacked is quite large, using a well designed non-exact algorithms that are fast and produce reasonably good solutions is an attractive or even the only option.

Nowadays, the distinctions between exact and non-exact algorithms are blurred as sometimes these techniques are merged to form a stronger hybrid. For example, in determining the bounding formula in Branch and Bound, people are usually using heuristic-like methods, that is, heuristic becomes part of another algorithm. We have also seen mergers between Constraint Programming and SLS algorithms and many others.

2.3 Stochastic Local Search (SLS) for Attacking COPs

2.3.1 Background

The two terms 'Metaheuristic (MH)'[14] and 'Stochastic Local Search (SLS)' are synonymous and interchangeable. In this thesis we adopt the term: 'SLS' made popular by [26].

This type of algorithm has been fast evolving in recent years. Since the pioneering works of meta-controller to escape local optima, e.g. simulated annealing [30], steepest descent mildest

whether we can get better results (e.g. 1.9 million $)?

[13]For example, the champion heuristic to date for TSP is the Lin-Kernighan (LK) heuristic [29]

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

ascent [??], Tabu Search [??], etc in mid 1980ies, there are many more SLS techniques that have been proposed. The list below highlights some of the popular ones:

- Tabu Search (TS) [??], [20], mustciteRayward Smith book
- Iterative Local Search (ILS) [46], [??]
- Simulated Annealing (SA) [30]
- Ant Colony Optimization (ACO) [??], [10]
- Genetic Algorithms (GA) [24]
- Memetic Algorithm (MA) [??], [34]
- Scatter Search (SS) [41]
- Variable Neighborhood Search (VNS) [22]
- Many others, including hybrids and/or variant approaches.

SLS algorithms are currently among the most efficient and robust optimization strategies to find high-quality solutions for many real-life COPs (see Chapter 2.1 and Appendix A). A large number of successful applications of SLS algorithms in various fields are reported in the literature: books [36, 1, 50, 4, 19, 37, 6, 26, 41, 12], journals, as well in scientific meetings or conferences, e.g. Metaheuristics International Conference (MIC series, bi-annually since 1995) [38, 51, 44, 43, 27, 9], Engineering SLS Workshop, CP, CP-AI-OR, INFORMS, European Chapter of Metaheuristics (EU/ME), etc, signifying a potential brightness of this field in the future.

2.3.2 What is SLS algorithm?

SLS can be described from various angles. The list below summarizes some different views of SLS. Some views that require further elaborations have specific subsections after this list.

1. SLS is a non-exact algorithm that can produce high quality solutions in a reasonable computing time. As it is a non-exact algorithm, an SLS algorithm will not stop upon encountering the optimal solution along the search process. It does not know that the solution is indeed optimal and will just continue searching until it reaches the terminating condition. Quality-wise, the solution qualities found by SLS are usually better compared to more simpler non-exact algorithms (approximation or heuristic algorithms) as it searches the fitness landscape rather than just zooming in into one solution.

2. Metaheuristic/SLS is a master procedure that iteratively guides and modifies the operations of subordinate heuristics (a simple local search, a construction method, or a simple greedy step, etc) to produce improved approximate solutions to $\mathcal{NP}$-hard COPs. SLS combines intelligent exploitation and exploration of the fitness landscape in order to find good solutions efficiently. Decisions of where to move are done by using local knowledge only, perhaps with some influence of randomness/stochasticity[15] to achieve more robustness [20]. The iterative process of SLS can be stopped at any time and the current best found solution BF is returned. This part is elaborated in Chapter 2.3.3.

3. Anatomically, SLS algorithm can be divided into two parts: algorithmic template M and configuration ϕ. The precise behavior and the performance of SLS algorithm is highly dependent on the chosen search parameters, components (heuristics), and search strategies. A more detailed explanation about this is in Chapter 2.3.4.

Additional Definitions for COP and SLS Algorithms

These definitions are assumed for a **minimizing** COP. They are summarized or adopted from various sources (e.g. [34, 26]) and enhanced with our own additional definitions:

Local Move : A small transformation from a current solution s into s'.

Improving Move : A local move where $g(s')$ is better than $g(s)$.

Non-Improving Move : A local move where $g(s')$ is worse than or equal to $g(s)$.

[15] In computer, true randomness cannot be truly achieved as computer can only generate 'pseudo random' numbers.

Neighborhood $N(\pi)$: The neighborhood operator (set of potential local moves) used by the SLS for traversing the search space $S(\pi)$. The choice of neighborhood is usually problem specific.

Distance Function $d(s_1, s_2) : S \times S \rightarrow \Re$: The distance between two solutions s_1, s_2 in an n-dimensional space is denoted as $d(s_1, s_2)$. This function is generic and the details are discussed in Appendix D.

Fitness Landscape FL : $\langle S(\pi), d(s_1, s_2), g(\pi) \rangle$. This is the space where the SLS operates. SLS can be seen as a walk on the neighborhood graph (see Figure 2.1 for an example). Fitness Landscape is a more general definition than Search Space. In this thesis, we use the term Fitness Landscape rather than Search Space. Note that in this thesis, we adopt the definition used in [34] (using distance function d) and not the one in [26] (using neighborhood operator N).

Local Optima/Minima LO : A solution s is said to be 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. Depending on the problem characteristic, the distance between LO may be near to each other (clustered) or far (sparse). $|LO|$ denotes the number of local optima for a particular COP instance. It is conjectured that if fitness landscape FL has many LO that are not GO, SLS will have more difficulties while searching on FL. Local Optima solutions are detrimental to SLS algorithms since they tend to make the SLS trapped within it.

Global Optima/Minima GO : A solution s is said to be global optima (optimal solution) if $g(GO)$ is the smallest among other $g(s'), s' \in S(\pi)$. Depending on the problem characteristic, there can be more than 1 GO solution for a COP instance π and the distance among the GO may be near or far.

Search Trajectory ST : ST begins with an initial solution s^0. At iteration t, ST moves from solution s^t to s^{t+1} and so on until the objective is reached or we have reached the limiting solution $s^{lastItr}$, i.e. $ST = s^0$ (initial solution) $\rightarrow s^1 \rightarrow \ldots \rightarrow s^{lastItr}$. This is a 'walk' in the fitness landscape FL. A good trajectory is usually those that intensify around good region and diversify when that good region is depleted from its potential. Efforts to escape a local optima can be the bottleneck for an SLS and therefore sometimes a good trajectory involves strong perturbation where s^t and s^{t+1} are reasonably different. $|ST|$ denotes the number of solutions along the search trajectory ST. In population based search, this terms become plural: search trajectories.

Best Found BF : A solution is said to be best found solution by an SLS algorithm if $g(BF)$ is the smallest among other $g(s'), s' \in ST$. BF may or may not be equal to GO.

Best Known BK : The best known solution for a COP instance, found either by exact algorithm or the best ever solution found by any SLS algorithm. Unless found by exact algorithm (proven to be optimal), the BK may or may not be the GO and called pseudo-optimal solution.

Region R : A connected subset of FL.

Plateau P : A region R in FL where each solution $s \in R$ has the same objective value. The definition of plateau can be generalized to region R with objective value $[OV-\epsilon \ldots OV+\epsilon]$ w.r.t to a point with objective value OV. This situation is commonly encountered by SLS while attacking COP with small range and/or have discrete objective values. During this situation, the SLS is like trying to find an island while being lost in a vast and flat seas without radar as it loses its main form of guidance: the objective value.

Ruggedness Rg : The reverse of plateaux landscape. Rugged landscape is also hard to traverse, mainly because it is now much more difficult to escape from a local optima with the SLS own escape mechanism.

Diameter DM : The longest walk from s to s' in the fitness landscape. The larger the neighborhood, the smaller the diameter is. Measuring the exact diameter is hard, but we can have loose upper bound of $O(n)$ where n is the size of π. It is conjectured that the smaller the diameter of a fitness landscape, the easier it is to search on that fitness landscape. An implication of this DM property is that from any solution/position in the fitness landscape FL, the SLS actually only needs at maximum DM steps to reach the optimal solution GO if it knows how to visit GO — nondeterministically.

Position P : In trajectory based search, the current solution can be seen as the 'current position' or 'point' in the fitness landscape. In population based search, this term is plural, as now position(s) refer to a set of current positions of each individual in the population.

2.3.3 Walks on COP Fitness Landscape

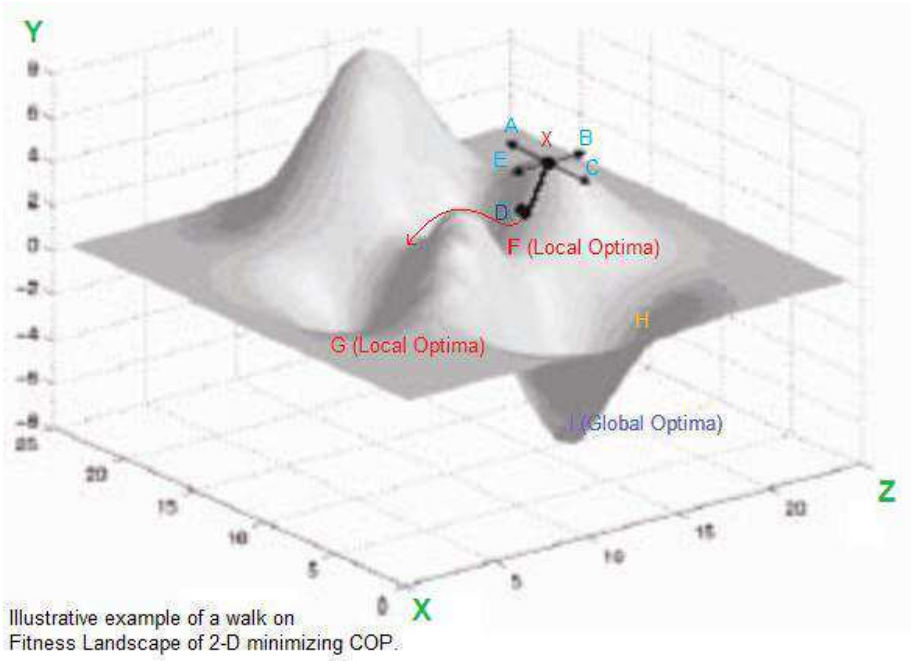

Figure 2.1: A Walk on Fitness Landscape. Legends: X-current solution; {A,B,C,D,E}-local neighbors of X; F and G-local optima; H-another solution; I-global optima. See text for details.

SLS algorithms can be abstractly described as follows. Consider an illustrative of example 2-D minimizing COP as shown in Figure 2.1, solution s for this COP is a possible instantiation of the COP decision variables (x-axis and z-axis). The set of all solutions $S(\pi)$ and their corresponding objective values $g(s)$ (y-axis) is called the *fitness landscape* FL and we also call a solution in the fitness landscape, a *point*.

In Figure 2.1, the letters A, B, C, D, E, F, G, H, I, and X denote solutions or points in the fitness landscape of our illustrative fitness landscape, each with its own objective value (the lower the better). Suppose the current solution is X. At this point of time, SLS algorithm determines a subset of the fitness landscape. In this case the local neighborhood of $N(X)$ is currently $\{A, B, C, D, E\}$. The greedy objective function heuristic selects a new solution from the neighborhood (e.g. improving move to point D) — an intensification. While objective function is a good guidance to steer the search, it may complicate the SLS if it makes the SLS trapped in local optima (e.g. F or G). Usually, with some stochastic probability, the SLS performs a jump to another solution outside the neighborhood, substantially far in terms of distance w.r.t to current position X (e.g. to position H where $d(X, H) > \epsilon$) — a diversification.

The red curve in Figure 2.1 is meant to illustrate a *search trajectory* ST – a series of solutions traversed during the search with the objective to search for the desired goal (e.g. I: the global optima GO). As SLS is incomplete, it may be trapped in local optima LO (e.g. F or G); or may exhibit cycling behavior (e.g. cycling among points X, D, E, X, D, E, ...). Upon termination, the SLS will report the best found BF solution that the SLS manages to encounter during its walk on the fitness landscape.

Each SLS differs in the way it traverses the fitness landscape. In general, an *ideal* SLS algorithm is the one that given a starting point in the COP fitness landscape, its walks exhibits these characteristics:

1. Navigate to the local optima of a region, intensifies around it.
2. Escape from any local optima region where it is currently in only after exploring the potential good local optima in that region.
3. Navigate to another local optima that is potentially closer to the true goal — the global optima, not searching in the 'wrong place'.

It is quite safe to say that if an SLS algorithm shows these characteristics within the allocated running time, it should have reasonably good performance, and hopefully upon termination, $BF == BK$ or $BF == GO$.

However, the behavior of SLS walk is often less than ideal. SLS relies on its *heuristic* to move locally within the fitness landscape (e.g. from current solution X to either A, B, C, D, or E). This mechanism is *incomplete* as it will not guarantee the SLS to reach global optimum I upon termination as local optima (e.g. F and G) may sway the SLS walk direction. SLS may use some form of *randomness* to help diversification (e.g. to H). However, the *stochastic* component also implies that the search trajectory ST may vary for different runs. These issues imply that making SLS performs well is not so straightforward.

Unlike systematic (exhaustive) search where one can predict the algorithm behavior, predicting SLS behavior is considerably harder in the sense that given the complete code of the SLS algorithm and a COP instance π, we cannot succinctly describe how the SLS will behave until we run it.

In the worst case, the algorithm designer may end up believing that the SLS is doing one thing while it is actually doing a different thing, which can be somewhat disastrous. For example, if an algorithm is supposed to do intensification but in fact it travels far from the original position in the fitness landscape, one cannot say that the intensification strategy is 'successful' or 'correct', regardless of whatever result that the algorithm managed to obtain. Even if the result is 'good', it is not because of the influence of the 'failed' strategy but may be due to another reason(s). This is coined as the metaheuristic/SLS 'failure modes' in [52].

2.3.4 Algorithmic Template M + Configuration ϕ

We say that SLS is an algorithm that have two major parts:

1. Fixed part (endogeneous), the **algorithmic template** M
   A basic skeleton of the algorithm. It is true that for each SLS M, there exists some variants of M' proposed by different researchers (e.g. a variant of TS that does not include AspirationCriteria at all), but we assume that the differences (if any) are minimal as an SLS algorithm will lose its identity and better renamed if its basic algorithm is altered too much.

   ```
   \SLS(initial-sol,configuration phi)
   \  current-sol = BF = initial-sol
   \
   \  while (termination-conditions-not-met)
   \    current-sol = best([neighborhood](current-sol,constraints,<parameters>))
   \    update data-structure
   \    if (current-sol is better than BF)
   \      BF = current-sol
   \    perform *search-strategy*
   \
   \  return BF
   ```

2. Tunable part (exogeneous), the **configuration** ϕ
   The selected configuration ϕ influences the behavior and thus the performance of the SLS algorithm. Configuration ϕ consists of three holistic parts and to find the correct ones to be matched with M is like solving a jigsaw puzzle:

   (a) Various set of **parameter values**
   Parameter values are adjustable numeric values within M which if set differently, have impact to the overall SLS performance. Usually experts in SLS can roughly gauge the possible range of these values after conducting some pilot experiments. Parameter values usually not constant but depends on the instance characteristics (e.g. the instance size) and it is quite sensitive towards SLS performance. These parameter values must be set before the SLS algorithm can run and if the values are not known, the user picks default/random values and conduct pilot experiments to find reasonable settings. Several parameter values are often correlated.

   (b) Many combination of **components**
   SLS components are essential parts of M that needs to be implemented or chosen to

make M works. Usually component is the place where the user places their sub-ordinate heuristics. Component can have some parameter(s) embedded within it. Typically, the choice of the best components are problem specific and not instance specific, thus not so sensitive while dealing with different instances. The choice of components are also often correlated, e.g. the pair of perturbation strategy and acceptance criteria in ILS. With parameters and components properly set, an SLS algorithm is ready to be executed and produce solutions.

(c) A lot of **search strategies**
Search Strategies are optional/additional codes on top of the M + {parameters + components} that are used to alter the behavior (trajectory) of the search, which ultimately to improve the performance of the SLS. Without these search strategies, the SLS algorithm will still works. However, it has been shown in the literature that the state-of-the-art heuristic algorithms are those which employ good search strategies. A search strategy is usually in form of contingent decision, that is, a certain action will be done if some criterion are met, e.g. the SLS algorithm will self-adjust its own parameter or component adaptively during the search after observing certain pattern.

Different choice of configuration ϕ will affect the way an SLS walks on various fitness landscape and also affect the performance of the SLS algorithm. Therefore, SLS algorithm designers like SLS with smaller configuration space ϕ because they are more user friendly for them. Algorithms that have more possible configurations are generally more difficult to tune.

The SLS algorithmic template M and configurations ϕ of several well-known state-of-the-art SLS algorithms used in this thesis are presented in Appendix B (last updated on: 25 July 2007!).

2.3.5 Implementation Issues

Unlike typical algorithm where rigorous mathematical proof is sufficient, to show the effectiveness of an SLS algorithm, we must implement, run the algorithm against COP instances, and obtain good solutions. Considering the heuristic and stochastic nature of the SLS, proper implementation of any 'good' SLS ideas is a must, otherwise they are just ideas without any guarantee that they will do what we think they will. SLS algorithm designer cannot escape from writing the basic algorithmic template M and the chosen configuration ϕ into a computer code.

Implementing SLS is usually a straightforward job but there are different ways to implement the same SLS algorithm, e.g. different data structure, coding skill, programming language, compiler, or computing platforms. These things will make the resulting performances different.

Fortunately, the algorithmic template M of typical SLS algorithms is common from one application to another and some components of the SLS are also frequently used. It is thus beneficial to apply software engineering principle of reusing these classes by creating SLS software frameworks or class libraries. This way, subsequent implementation of the SLS can be much faster because one can just reuse his previous codes[16]. It is worth mentioning that there are several SLS software frameworks available in the literature, each has their own strengths and weaknesses, e.g. EASYLOCAL++ [8], OpenTS [23], Localizer++ [35], HotFrame [15], Heuristic Search Builder (HSB) [11], COMET [49], Tabu Search Framework (TSF) [32], and Metaheuristics Software Framework (MDF) [33, 31][17].

A Good Way to Implement SLS Algorithm

There is a clever technique to avoid recompiling the source code many times if the SLS algorithm needs to be tweaked. This may save a lot of development time. Basically, the M part is hard coded but the ϕ parts are not. One can use a simple external text-based configuration file that is passed to the SLS executable (e.g. via command line parameters). This configuration file contains the following information for each parts of the configuration:

1. Numeric parameter values — straightforward

[16] However, to gain the ultimate performance, the algorithm designer cannot rely on software frameworks. Software frameworks boast their generic feature but at the same time they impose some overheads. We can have a better performing implementation if we implement the same SLS algorithm from scratch, then optimize the code manually.

[17] MDF is our work too, but it is outside the scope of this PhD thesis.

2. Options to choose SLS components — use if-else statements inside the SLS code to activate the corresponding SLS component
3. Options to turn on/off certain search strategies — use if-else statements to call/not to call a search strategy

The SLS implementation can also log the runtime data into standard text file for further analysis, as shown in Chapter 4, Chapter 6, and Appendix E.

2.3.6 Performance Evaluation

SLS is Lacking Theoretical Results

Analyzing SLS performance theoretically is not easy. Usually we do not have any theory for a particular SLS. And when we have, the theories are mainly worst case analysis and not average case analysis since we do not know the probability distribution of each case. Thus the information gained is not that practical as it is only applicable under very restrictive assumptions. A notable example is the proof of convergence for Simulated Annealing SLS algorithm [**??**]. It is said that SA will converge given infinite (∞) time. This is an interesting proof but impractical.

While discussing theoretical results of optimization algorithm, we cannot forget the 'No Free Lunch' (NFL) theorem by Wolpert and Macready [54], see Figure 2.2. This theorem basically said that there is no single algorithm that works best for *all instances* of the problem. A *conservation of competence* principle applies: the better one algorithm in solving one specific instance (class) the worst it is solving a different instance (class). Thus, "without restricting the class of problems considered in the scientific research, it is impossible to predict the performance of algorithms on real-life COPs".

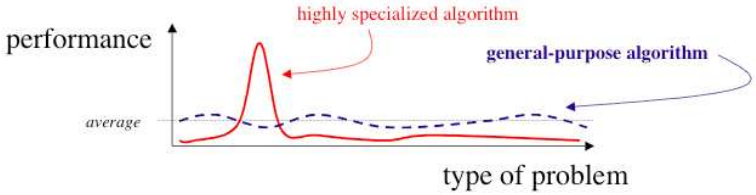

Figure 2.2: An illustration of the No Free Lunch Theorem

This theorem suggest that on the average, no single SLS algorithm (TS, ILS, SA, ACO, GA, etc) is better than random search. Success comes from adapting the technique to the problem at hand. To have a good solver for a given COP, one should produce specialized algorithms where each algorithm tackle specific (subset) of instances of the problem, exploiting problem-specific information as much as possible while avoiding over-fitting.

Evaluating SLS Performance via Empirical Analysis

Unlike exact methods for a given problem which mainly compete with other exact methods in term of speed, the decision of whether a particular heuristic based strategy perform well or not is more difficult. We may need to devise various other performance evaluation metrics, e.g. solution quality, consistency, etc.

As mentioned before, SLS are very hard to be evaluated theoretically. Currently, SLS algorithms are analyzed empirically, either via statistical techniques or via information visualization. Such empirical analysis must be done properly, as highlighted in the following papers in the literature [2, 25, 40, 13, 14, 17]. We provide the following simple example for illustration.

Suppose we run two SLS algorithms A and B to 4 different instances of a minimizing COP and we got these results:

A: (30, 20, 20, 20), mean $M = 22.5$, median $Med = 20$, standard deviation $s = 5$
B: (23, 23, 23, 23), mean $M = 23$, median $Med = 23$, standard deviation $s = 0$

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

Issue 2 - 'outliers':
Now if we omit the first instance, then the 'apparent' performance will be as follows:
A: (20, 20, 20), mean M: 20, standard deviation s: 0
B: (23, 23, 23), mean M: 23, standard deviation s: 0
This highlights that the selection of test instances also play a role in examining or comparing SLS algorithms.

Issue 3 - is it significant? - inferential statistic:
As we run SLS algorithms A and B on sample of 4 COP instances, we are interested to know whether our results is can be generalized to a population of COP instances. This is the job of inferential statistic and often this part is omitted in SLS literature. The choice of parametric versus non-parametric tests may also affect the results. For example, at $\alpha = 5\%$, parametric test of Student paired t-test fails to reject the null hypothesis that the difference between SLS A results and SLS B results is significant, whereas the non-parametric counterpart of Wilcoxon signed rank test rejects the null hypothesis with the same data. For details about statistical techniques, refer to books like [21], [**??**].

2.4 Summary

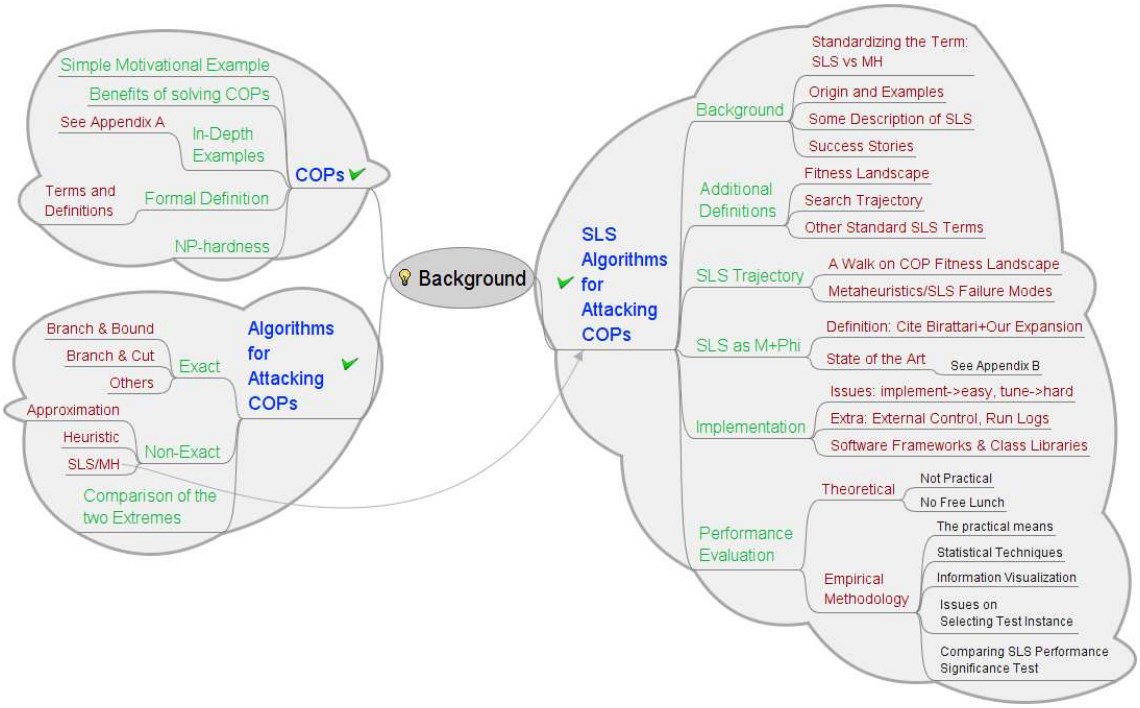

Figure 2.3: Chapter Overview

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

2. There are two extremes for attacking $\mathcal{NP}$-hard COPs: exact and non-exact algorithms, each with their own pros and cons. In this thesis, we focus on the non-exact algorithm called Stochastic Local Search (SLS), also known as metaheuristic.

3. We have presented various aspects of SLS algorithms: background, various views of what is SLS, SLS as walks on COP fitness landscapes, SLS as two components: $M + \phi$, SLS implementation, and SLS performance evaluation.

2.5 Further Discussions

The existence of SLS algorithms does not automatically solve the difficulty of attacking $\mathcal{NP}$-hard COPs as the performance of SLS algorithms depends a lot of the chosen $M + \phi$. The users still need to mix and match to find the correct ϕ for their chosen M. There is 'no free lunch', we need to adapt SLS techniques to the COP at hand to obtain good performance!

Previously, the process of choosing the appropriate $M + \phi$ is still more like an art rather than science. There are various suggestions in the literature but which one to follow? There are also successful reports for certain COP A or B, but are the results applicable for my own COP C, even if COP C is just a variant of COP A? Can we have a bit more systematic way for dealing with new problem/new variant of classical COP?

In Chapter 16 of [19], it is said that while developments in SLS have deepened our scientific understanding of the search process and the automatic solution of large complex problem, it is true to say that the practical impact in commercial and industrial organizations has not been as great as might have been expected some years ago. Many state-of-the-art SLS developments are too problem-specific or knowledge-intensive to be implemented in cheap, easy-to-use computer system. What can be done about this?

These issues gives rise to the main problem that are going to be addressed in this thesis, the SLS Design and Tuning Problem. We discuss it in Chapter 3.

Note: more examples of classic COPs and popular SLS algorithms are listed in Appendix A and B, respectively.

Bibliography

[1] Emile Aarts and Jan Karel Lenstra. *Local Search in Combinatorial Optimization*. John Wiley and Sons, 1997.

[2] 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.

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

[4] Christian Blum and Andrea Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35:268–308, 2003.

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

[6] Edmund K. Burke and Graham Kendall. *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*. Springer, 2005.

[7] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, second edition, 2001.

[8] 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.

[9] Karl F. Doerner, Michel Gendreau, Peter Greistorfer, Walter J. Gutjahr, Richard F. Hartl, and Marc Reimann. *Metaheuristics: Progress in Complex Systems Optimization*. Springer, 2007.

[10] Marco Dorigo and Thomas Stützle. *Ants Colony Optimization*. MIT Press, 2004.

[11] Raphaël Dorne, Cedric Ladde, and Christos Voudouris. Heuristic Search Builder The iOpt's Tool to visually build metaheuristics algorithms. In *4th Metaheuristics International Conference*, 2003.

[12] Johann Dreo, Alain Petrowski, Patrick Siarry, and Éric D. Taillard. *Metaheuristics for Hard Optimization: Methods and Case Studies*. Springer, first edition, 2006.

[13] Ágoston E. Eiben and Márk Jelasity. A Critical Note on Experimental Research Methodology in EC. In *IEEE International Conference on Evolutionary Computation*, pages 582–587, 2002.

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

[15] Andreas Fink and Stefan Voß. HotFrame: A Heuristic Optimization Framework. In *Optimization Software Class Libraries*, pages 81–154. Kluwer Academic Publishers, 2002.

[16] Michael R. Garey and David S. Johnson. *Computer and Intractability: A Guide to the Theory of NP-Completeness*. William H. Freeman, 1979.

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

[18] Faruk Geyik and Ismail Hakki Cedimoglu. The strategies and parameters of tabu search for job-shop scheduling. *Journal of Intelligent Manufacturing*, 15:439–448, 2004.

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

[20] Fred Glover and Manuel Laguna. *Tabu Search*. Kluwer Academic Publishers, 1997.

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

[22] Pierre Hansen and Nenad Mladenovic. A Tutorial on Variable Neighborhood Search. *TR G-2003-46, GERAD*, 2003.

[23] Robert Harder. OpenTS. *IBM OpenTS Homepage: http://opents.iharder.net (online)*, 2001.

[24] John Henry Holland. *Adaptation in Natural and Artificial Ecosystems*. MIT Press, second edition, 1992.

[25] John N. Hooker. Testing Heuristics: We Have It All Wrong. *Journal of Heuristics*, 1:33–42, 1995.

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

[27] 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.

[28] ILOG. ILOG.
http://www.ilog.com.

[29] 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.

[30] Scott Kirkpatrick, D. Gelatt Jr, and Mario P. Vecchi. Optimization by Simulated Annealing. *Science*, 220:671–680, 1983.

[31] 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.

[32] Hoong Chuin Lau, Wee Chong Wan, and Xiaomin Jia. A Generic Object-Oriented Tabu Search Framework. In *5th Metaheuristics International Conference*, 2003.

[33] 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.

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

[35] 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.

[36] 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.

[37] 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.

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

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

[40] Ronald L. Rardin and Reha Uzsoy. Experimental Evaluation of Heuristic Optimization Algorithms: A Tutorial. *Journal of Heuristics*, 7(3):261–304, 2001.

[41] César Rego and Bahram Alidaee. *Tabu Search and Scatter Search*. Kluwer Academic Publishers, 2005.

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

[43] Mauricio G.C. Resende and Jorge Pinho de Sousa. *Metaheuristics: Computer Decision-Making*. Kluwer Academic Publishers, series=Applied Optimization, volume=86, year=2003.

[44] Celso C. Ribeiro and Pierre Hansen. *Essays and Surveys in Metaheuristics*, volume 15 of *Operations Research/Computer Science Interfaces*. Kluwer Academic Publishers, 2001.

[45] Marius M. Solomon. Algorithms for Vehicle Routing and Scheduling Problem with Time Window Constraints. *Operations Research*, 35:254–265, 1987.

[46] 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.

[47] Éric D. Taillard. Robust Tabu Search for Quadratic Assignment Problem. *Parallel Computing*, 17:443–455, 1991.

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

[49] Pascal van Hentenryck and Laurent Michel. *Constraint-Based Local Search*. MIT Press, Cambridge, London, 2005.

[50] 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.

[51] 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.

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

[53] Wayne L. Winston. *Operations Research: Applications and Algorithms*. Thomson Brooks/Cole, 2004.

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

Index