TU Clausthal

Institut für Angewandte Stochastik und Operations Research

Model-based heuristics for combinatorial optimization: a mathematical study of their asymptotic behavior

PhD Thesis

January 4, 2015

Author:
Zijun Wu

Supervisor:
Prof. Dr. Michael Kolonko

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgement.

Signed:
Date:

Acknowledgement

First, I want to sincerely thank my supervisor Mr Prof. Dr. Michael Kolonko. He gave me much support in my study and led me to such an interesting field. And I also want to thank him for agreeing me to collect our co-work in this Thesis.

Then, I want to thank my colleagues Mr Dipl.-inf Stephan Mock, Mr M.Sc Zhixing Yang, Mr Dipl.-Math. Fabian Kirchnoff, Miss Gesine Redecker and Miss Baerbel Heise-Kretzer, they gave me numerous help during my stay in Clausthal.

I also want to thank my wife. Without her understanding and support, I would not have been able to concentrate on my study.

Special thank goes to the Chinese Government, they financially supported my study from December 2010 to November 2014.

Abstract

Due to the complexity of many important combinatorial optimization problems, heuristic search algorithms are of overwhelming importance for a practical solution of many problems in Operations Research like tour planing, vehicle routing, scheduling, packing etc. Many traditional heuristic procedures are 'solution-based', e.g. tabu search, simulated annealing, genetic algorithms etc. Here, new solutions are produced trough manipulating current solutions with a specified strategy like edge exchanging in tour planning, crossover and mutation in genetic algorithms etc.

Recently, a different class of algorithms have gained attention which do not concentrate on solutions, but on the mechanism to produce them. The mechanism is typically a distribution on the space of solutions. It is often called a 'model' for the solution space, and new solutions are produced by sampling from it. Starting from a fixed initial model, these algorithms then iteratively evolve the present model by adapting it to some 'best' solutions sampled from the present model and/or previous models. By the evolution, they expect to reach a model concentrating only on some optimal solutions. Examples of these 'model-based' heuristic procedures include ant colony optimization algorithms, cross entropy algorithms and estimation of distribution algorithms.

This Thesis concentrates on these model-based heuristics i.e. model-based search. We propose a framework which covers the essential features of these algorithms in practice. The framework contains a new concept which can include probabilistic dependencies into the sampling. We study closely the resulting stochastic process of the framework. We state simple conditions which guarantee to reach an optimal solution in finitely many iterations. For some standard test problems, we show conditions which imply a (low-degree) polynomial runtime with a probability converging to 1 as the problem size approaches infinity.

We also investigate the asymptotic properties of the samples and models. We show that the sampling may get frozen at a fixed solution after finitely many iterations, and the models may converge to one-point measure. This theoretically proves the stagnation behavior observed in the literature. In particular, we propose conditions which make the models converge to a limit concentrating only on optimal solutions.

We complement the theoretical analysis with a computational study on the famous traveling salesman problem. The experimental results clearly demonstrate our theoretical findings, and show some useful hints for a practice use.

Keywords:
combinatorial optimization; heuristics; ant colony optimization; cross entropy; estimation of distribution algorithms; distribution learning; convergence; runtime analysis; genetic drift; unsupervised learning; stochastic process.

Acronyms

$\mathcal{MMAS}$ $\mathcal{MAX}$-$\mathcal{MIN}$ ant system. 31

ACO ant colony optimization. 23

ACS ant colony system. 30

AP assignment problem. 9

AS ant system. 26

CE cross entropy algorithm. 16

cGA compact genetic algorithm. 43

CO combinatorial optimization. 6

COP combinatorial optimization problem. 8

EDA estimation of distribution algorithms. 37

FIFO first in first out. 59

GTMU globally truncate memory update. 59

ID identity selection. 60

KP knapsack problem. 9

LTMU locally truncate memory update. 59

MaxCut maximal cut problem. 9

MBS model-based search. 11

MID memory identical selection. 60

MRS memory random selection. 60

NM non memory. 59

PBAS population-based ant system. 31, 59

PBIL population-based incremental learning. 41

RES rare event simulation. 149

RS random selection. 60

SBS solution-based search. 11

TDWL time dependent weighted learning. 61

TS truncate selection. 60

TSP traveling salesman problem. 8

UL uniform learning. 60

UMDA univariate marginal distribution algorithm. 40

WL weighted learning. 60, 61

WO worst out. 59

Contents

List of Figures

List of Tables

List of Symbols

$C_i(\cdot;\cdot)$ the priori feasibility distribution for position $i+1$. 51

$C_i(y)$ the support of feasibility distribution $C_i(y;\cdot)$. 51

L the encoded solution length in a string encoding. 16

$Q(a; y, i+1, \mathbf{\Pi})$ the selection probability for alphabet a at position $i+1$ when the leading partial solution is $y \in R_i$ in a feasibility construction. 52

$Q\big((s_1, \ldots, s_L); \mathbf{\Pi}\big)$ the selection probability for a solution $(s_1, \ldots, s_L)$ in a feasibility construction. 52

R_i the collection of partial solutions with length i. 51

S^* set of optimal solutions. 6

S feasible solutions. 6

$\mathbb{1}_A$ indicator function of a set A. 9

$\mathcal{A}$ the finite alphabet for an instance in string encoding. 16

$\mathcal{L}$ learning rale. 63

$\mathbf{M}_t$ the memory for iteration t. 57

$\mathcal{M}$ memory update rule. 63

$\mathbb{N}$ natural numbers. 17

$\mathcal{N}_t^b$ the selected subsample from $\mathbf{M}_t \cup \mathbf{X}_t$. 57

$\diamond$ the empty string. 47

$\mathbf{\Pi}(a; i)$ the probability of alphabet a occurring at position i under distribution $\mathbf{\Pi}$. 17

$\mathbf{\Pi}_t$ the model in iteration t. 15

$\mathbb{R}_+$ set of positive real numbers. 17

$\mathbb{R}$ set of real numbers. 6

$\mathcal{S}$ selection rule. 63

$\mathbf{W}_t$ the empirical model for iteration t learned from $\mathcal{N}_t^b$. 18

$\mathbf{X}_t$ the random sample in iteration t. 15

$\in$ the belonging relation. 8

f objective function, cost function. 6

1 Introduction

Combinatorial optimization [KV02] is a very important topic in applied mathematics and theoretical computer science. It can be simply stated as finding out an object from a finite collection of candidates which minimizes (or maximizes) an associated objective function. A well-known problem involving combinatorial optimization is the traveling salesman problem (TSP), which concerns finding a shortest tour that starts in a city, traverses each other city exactly once, and finally returns to the start city. Actually, combinatorial optimization can be seen almost everywhere in reality. In logistics, we may need to determine routes with a minimal total delivering cost for a fleet of vehicles which are employed to deliver goods from a central depot to consumers who ordered the goods. In a water supply system, we may need to place several valves at pipes' extremes such that when a pipe breaks, it is possible to isolate it with minimal damage to the rest of the network. In a factory, we may need to arrange some jobs on some machines such that the resulting makespan becomes minimal. In a harbor, we may need to determine a minimal number of ships for loading certain goods which are of different volumes and shapes.

In combinatorial optimization, we often call the collection of candidates a feasible set, an object belonging to the collection a feasible solution, and a feasible solution minimizing (or maximizing) the associated objective function an optimal solution. Here, the objective is to find an optimal solution from the finite feasible set. In mathematical optimization, combinatorial optimization is usually categorized as a subtopic of discrete optimization. It is closely related to operations research, algorithm theory and computational complexity theory. It has important applications in other fields like transportation, machine learning, artificial intelligence, software engineering, job shop management, information and communication techniques etc.

Due to the discrete nature, generally we can not get an analytic solution for a combinatorial optimization problem. Thanks to the dramatically increased computational capacity in recent years, we now can exactly solve many combinatorial optimization problems under a moderate size by running an exact search [Woe03] on a computer, e.g. we may enumerate all feasible solutions. However, as the problem size increases, the complexity of the problems may increase exponentially. When the problem size is relatively large, we may not get an exact optimal solution in an acceptable run time. To quickly obtain a practical solution in the case of a large problem size, we have to employ some non-exact search procedures. Generally, these procedures can be collected in two classes: approximation algorithms [WS11] and heuristics [Nic07]. Approximation algorithms are those which are designed according to some particular features of the underlying problem, and guarantee to find a high-quality approximation for the optimal solutions. Typically, they can guarantee that the approximation is optimal up to

a small constant factor, say 5%. However, they may depend heavily on the structure of the underlying problem, therefore can not be used for a general-purpose.

Heuristics do not guarantee on solutions' qualities. However, many applications have demonstrated that they can reach high-quality solutions in most cases. They are often equipped with some 'intelligent' or 'self-adaptive' mechanisms. These mechanisms are used to guide the underlying search in the feasible set. But, these mechanisms are generally designed without involving much prior knowledge about the underlying problem. This makes heuristics rather portable in practice. So, heuristics can be seen as general-purpose tools for optimization. As a general-purpose tool, heuristics can be used in many other fields, e.g. bioinformatics, machine learning, artificial intelligence.

Heuristics can be collected in two subclasses: solution-based heuristics (search) and model-based heuristics (search), see [ZBMD04]. Solution-based search concentrates on solutions. They iteratively evolve solutions. New solutions are produced by manipulating present solutions in a way that we can expect a better solution in the next round. Many traditional heuristics are solution-based. Examples are these nature-like algorithms as tabu search [GL99], simulated annealing [BT93], genetic algorithms [Mic96] etc. Here, new solutions are produced by changing and/or exchanging components on present solutions such that new solutions will 'inherit' good properties of present solutions. For example, when we apply a genetic algorithm to a TSP, new tours (solutions) are produced by performing crossover (exchanging edges on two selected parent tours) and mutation (randomly changing one edge on a child) on the present tours. More recent examples of solution-based search are some swarm intelligence algorithms, including artificial bee colony [KGOK12], particle swarm optimization [KE95] and electromagnetism-like algorithm [BF03] etc. They aim to simulate the 'collective' behavior in a school of practical creatures. They iteratively evolve a swarm of solutions. The evolution of each individual (solution) in the swarm may refer to both its own history and the histories of its companions. For example, in a particle swarm optimization, an individual is evolved with reference to both its own best experience and the global best experience accumulated by the whole swarm.

Model-based search does not concentrate on solutions, but on the mechanism to produce solutions. This mechanism often explains the structure of the underlying problem to some extent. It is therefore called a 'model' of the problem or its solutions. Typically, it is a probabilistic distribution on the feasible set. Here, new solutions are produced by sampling from a present model. Model-based search iteratively evolves models instead of solutions. By evolution, they aim to reach a model which can produce optimal solutions only or with an overwhelming probability. Examples of model-based search are cross entropy algorithms [RK04], ant colony optimization algorithms [DS04], and estimation of distribution algorithms [HP11]. Starting from an arbitrarily fixed initial model $\mathbf{\Pi}_0$, model-based search then iteratively evolve models as, for $t = 0, 1, 2, \ldots,$

Sampling: generate a random *sample* $\mathbf{X}_t$ of a specified size $N \in \mathbb{N}$ by the present model $\mathbf{\Pi}_t$;

Learning: learn an *empirical model* $\mathbf{W}_t$ from a *subsample* $\mathcal{N}_t^b$ consisting of some selected 'best' solutions which are seen in present sample $\mathbf{X}_t$ and/or in history;

Update: set the next model $\mathbf{\Pi}_{t+1} = (1-\rho_{t+1})\mathbf{\Pi}_t + \rho_{t+1}\mathbf{W}_t$, where $\rho_{t+1} \in (0,1]$ is a *learning rate* fixed in advance.

In combinatorial optimization, the feasible set is finite. A model (or a distribution) can be therefore described by a vector. So, the next model $\mathbf{\Pi}_{t+1}$ is actually a convex combination of present model $\mathbf{\Pi}_t$ and the learned empirical model $\mathbf{W}_t$, where the learning rate ρ_{t+1} reflects the relative importance of $\mathbf{W}_t$ in the combination. The subsample $\mathcal{N}_t^b$ typically consists of some best solutions seen in $\mathbf{X}_t$ and history, for example, it may consist only of best solution found so far or of some elite solutions in present sample $\mathbf{X}_t$. By learning from $\mathcal{N}_t^b$, the empirical model $\mathbf{W}_t$ may contain information about good solutions. By the combination, the next model $\mathbf{\Pi}_{t+1}$ is therefore biased towards good solutions. Thereby, we may expect that in the next sampling, more good solutions will be produced. By iteratively evolving through the three steps, we hope that the resulting models process $\left(\mathbf{\Pi}_t\right)_{t=0,1,2,\ldots}$ can converge to a limit $\mathbf{\Pi}_\infty$ concentrating on optimal solutions i.e. the probability for producing optimal solutions by $\mathbf{\Pi}_\infty$ is 1. So, in model-based search, we actually 'optimize' the mechanism for producing (optimal) solutions.

Model-based search may also apply to other fields. Typically, they can apply to rare event simulation which usually concerns calculating the probability of a rare event in a complex system, see [RT$^+$09] or A. 6 in Appendix. Due to the rarity of the event, the classic monte-carlo method may fail to give an effective approximation to that probability. Here, a well-known approach is to employ importance sampling under a *best* change of measure selected from a specified family of candidate measures (distribution or density). Since model-based search evolve models (distributions), they can be applied easily to select the best change of measure. A successful example is the cross entropy algorithm for rare event simulation, see [Rub97] and [Rub99].

Model-based search are closely related to other fields involving estimation of distribution or density. A typical example is the so-called unsupervised learning in the field of machine learning [Alp10]. In unsupervised learning, we want to detect the regularities concealed in the input data. Typically, we may build a multivariate distribution (or a density) which fits the input observations best. By this distribution, we can determine the mutual dependencies of different variables. This may coincide with the 'Learning' step in model-based search. They can therefore share the learning method. A method used in the 'Learning' step may also apply to unsupervised learning, and vice versa. For example, the Bayesian network is used in both Bayesian optimization algorithm [Pel05] and the unsupervised learning.

This Thesis concentrates on model-based search. We will inspect their long-term behavior in combinatorial optimization. To do this, we will build a framework which can cover the essential features of these model-based search algorithms used in practice. The framework will result in a mixed Markov process, i.e. some marginal processes are discrete and other marginals are continuous. We will do a thorough mathematical analysis of this mixed process. Special emphasis will be given to the marginal processes formed by samples $\mathbf{X}_t$ and models $\mathbf{\Pi}_t$. Due to the non-homogeneity and complexity of the mixed process, we will not go along with the classical analysis for Markov chain. Our

analysis will involve only the memory-less property of the Markov chain. However, some basic knowledge about probability theory is required. Readers who are not familiar with probability theory can see A. 4 in the Appendix as a reference.

We will launch the analysis by stating simple conditions for guaranteeing to reach an optimal solution. These conditions should be of great interests in practice. In [Gut00] and [Gut03], W. J. Gutjahr showed that for a particular ant colony optimization algorithm, we can increase the occurring probability of optimal solutions by decreasing the evaporation rate (learning rate) or increasing the sample size N. In [CJK07], A. Costa et al inspected some important asymptotic properties of a generalized cross entropy algorithm for unconstrained combinatorial optimization problems. They showed conditions guaranteeing to reach an optimal solution. In our former work [WK14b], my supervisor and I inspected a more general cross entropy optimization algorithm which also covers the essential features of some ant colony optimization algorithms. In that work, we did not impose any restriction on the underlying problem. Still, we are able to show that the conditions proposed in [CJK07] hold in the more general algorithm. In Chapter 5, we will continue the study of our former work. We show that the findings in [Gut00], [Gut03], [CJK07] and [WK14b] also hold in our more general framework, see Theorem 5.7. Therefore, they may apply to all algorithms covered by the framework. In particular, we find that for the popular case $\rho_t \equiv \rho > 0$ used in practice, optimal solutions may not occur.

In recent years, the runtime analysis for heuristics has become a very popular field, see e.g. [NW06], [NW09], [DNSW07], [DJ07], [Gut07], [Gut08], [CTCY10] and [WK14b]. In runtime analysis, we want to find conditions which make an algorithm reach an optimal solution efficiently with a high probability. Here, the runtime is a rudimentary copy of the computational complexity in theoretical computer science. It is generally defined as the total number of solutions evaluated before reaching an optimal solution. In [NW06], [NW09], [DNSW07], [DJ07], [Gut07], [Gut08], researchers considered runtime for ant colony optimization algorithms with restricted models on some simple test problems like OneMax and LeadingOne. They found that the runtime is closely related to the learning rate, and we may reach an optimal solution efficiently by adapting a constant learning rate to the problem size if we use restricted models. In [CTCY10], Chen et al initiated a different study for the case of non-restricted models. They inspected the runtime for a univariate marginal distribution algorithm (a particular cross entropy algorithm with $\rho_t \equiv \rho = 1$). They showed that in the case of non-restricted models, we may also reach an optimal solution efficiently by adapting the sample size to problem size. Our former work [WK14b] extends the finding in [CTCY10]. We showed that actually for the more general case $\rho_t \equiv \rho > 0$, the finding in [CTCY10] may still hold. In Chapter 5, we will collect our former runtime results in Theorem 5.8. Moreover, we will propose a new runtime result in Theorem 5.9 for the case of restricted models.

It is not surprising that solutions' qualities may stop improving after finitely many iterations for heuristic algorithms in combinatorial optimization, because of the finite size of the feasible set. However, in model-based search, the reason is not so simple. [DMC96] and [DBKMR05] observed a phenomenon that after finitely many iterations, the 'Sampling' in ant system and cross entropy algorithm may be frozen at a fixed

solution. In other word, the algorithms may completely lose randomness after finitely many iterations. This coincides with the well-known 'genetic drift' phenomenon [AM94] in genetic algorithms. In our former work [WK14b] and [WK14a], we showed for cross entropy algorithm and a more general algorithm, resp., that when the learning rates $\rho_t \geq \rho > 0$ for each $t \in \mathbb{N}$, the phenomenon would occur with probability 1. In Chapter 6, we formally define the phenomenon as absorption of solutions. We will show that absorption of solutions still holds in our framework if the learning rates $\rho_t \geq \rho > 0$ for each $t \in \mathbb{N}$ and constant ρ, see Theorem 6.1. Therefore, model-based search may keep search ability only in finitely many iterations. After that, they may become deterministic and stick on a fixed solution. In particular, we show that when absorption of solutions holds, optimal solutions may not occur, see Theorem 6.4. Moreover, we find that the solution which freezes the sampling is typically an iteration-best solution or best found solution in the search history, see Theorem 6.3. Here, it is worthy to mention that inspired by the proof of Theorem 6.1, we are able to formalize a simple method which may be helpful in quickly determining some asymptotic properties for a non-homogeneous Markov chain.

The asymptotic behavior for the models is of great importance in model-based search. We are eager to know the conditions which make the models converge to a limit concentrating on optimal solutions. In [Gut02] and [Mar05], researchers showed conditions for some particular algorithms which learn the empirical model only from the best solution found so far. In [WK14b], we showed that in cross entropy algorithm, convergence of models and occurrence of optimal solutions are compatible. This settles an open question proposed in [CJK07]. In [ZM04], Zhang et al showed that the models in univariate marginal distribution algorithm will converge to a limit concentrating on optimal solutions, if we assume an infinite sample size. In Chapter 6, we continue the research in [Gut02], [Mar05] and [WK14b]. We show that if absorption of solutions holds, the models will converge to a limit which concentrates on a single solution, see Theorem 6.5. Moreover, we are able to show conditions which make models converge to a limit concentrating on an optimal solution, see Theorems 6.6-6.7. The conditions here greatly weaken the conditions proposed in [Gut02] and [Mar05].

The remaining of the Thesis consists of 7 Chapters. As a background, we will introduce combinatorial optimization and heuristics in Chapter 2, and make a short tutorial for these model-based algorithms used in practice in Chapter 3. In Chapter 4, we will propose a unified framework based on these algorithms in practice, and summarize the common rules used by them. Our theoretical findings will be proposed in Chapter 5 and Chapter 6. In Chapter 5, we will report the results related to occurrence of optimal solutions. In Chapter 6, we will report the results on asymptotic properties of solutions and models. In Chapter 7, we complement the theoretical analysis with a computational study on a TSP instance. The experimental results will clearly demonstrate the theoretical findings, and show some useful hints for a practical use. In Chapter 8, we will give some suggestions for a future research and make a summary to the whole Thesis. Some useful backgrounding knowledge are collected in the Appendix.

2 Background: combinatorial optimization and heuristic search

This Chapter serves as a background of our main topic model-based search (heuristics). We will review some basic elements for combinatorial optimization and give a short introduction to heuristic search. Readers who are very familiar with them can skip over this Chapter. The whole Chapter is arranged as: Section 2.1 defines several frequently used elements in combinatorial optimization; Section 2.2 shows some benchmark examples of combinatorial optimization, which will be frequently referred in the sequel; Section 2.3 gives a brief introduction to heuristic search with emphasis on its classification.

2.1 Combinatorial optimization: some elements

Optimization or mathematical optimization is a study subject which concerns finding an optimum for a certain function in a given (restricted) domain, see [GL95]. According to an encoding of the domain, we may collect optimization into two categories: continuous optimization and discrete optimization, see [Gou06] p. 1. Generally, discrete optimization concerns a search in a countable collection, i.e. the domain is encoded discretely, see [BR03]. Here, we only talk about combinatorial optimization, a subtopic of discrete optimization. Readers who are interested in continuous optimization, see [AEP05] for a reference.

Combinatorial optimization (CO) is a very important and popular subtopic of discrete optimization. It has applications in various fields, e.g. machine learning, artificial intelligence, software engineering, job shop management, information and communication techniques. According to [Law01] pp. 1-2, CO concerns finding an optimal arrangement, grouping, ordering or selection from finitely many candidates. Here, we consider CO more extensively as a finite optimization, i.e. a study subject of searching for an optimal object within a finite collection of candidates.

Let S be a *non-empty* finite set, f a real function on S i.e. $f : S \mapsto \mathbb{R}$, and $\mathcal{O} \in \{0, 1\}$ a constant. Then the triple $(S, f, \mathcal{O})$ is called a CO *instance.* And if $\mathcal{O} = 1$, we say further that $(S, f, \mathcal{O})$ is a *maximizing instance*, otherwise it is called as a *minimizing instance.*

Let $(S, f, \mathcal{O})$ be a CO instance. We call each $s \in S$ a *feasible solution*, S the *feasible set* and f the *objective function* of that instance. For each $s \in S$, $f(s)$ is called the *objective value* of s. If it is a minimizing instance (i.e. $\mathcal{O} = 0$), then an *optimal solution* is defined as a feasible solution which minimizes the objective function f. Otherwise, an optimal solution is a feasible solution which maximizes f. Throughout this thesis, we shall denote the collection of optimal solutions as S^*. Obviously, $S^* \subseteq S$. For each

$s^* \in S^*$, its objective value will equal the same value and we call this value the *optimum* of that instance. These definitions shall be frequently used in the sequel. To further understand them, some examples are collected in Section 2.2.

Given a CO instance $(S, f, \mathcal{O})$, the *objective* of CO is to find out an optimal solution $s^* \in S^*$ for this instance. Due to the finite size of the feasible set S, it must exist an optimal solution i.e. $|S^*| \geq 1$. In theory, we can find an optimal solution by enumerating all possible candidates in S. However, this is often infeasible in practice, especially when the size of S is extremely large. The main reason is that enumerating a huge collection may require not only a huge memory, but also a prohibitive time. Although the capacity (including computational speed and memory) of computers has been dramatically improved in these years, it is still too limited compared to the practical demand in CO. Therefore, we need resort to some more 'clever' search procedures. Among them, the procedures of model-based type have gained numerous attention in these years. This Thesis will concentrate on these procedures, and aim to inspect their asymptotic properties in CO.

2.2 Combinatorial optimization: examples

In CO, an optimal solution to a particular instance is not of much interests. We are more interested in a unified method to solve a class of instances of a certain type. Formally, a specified class of CO instances is often called a *combinatorial optimization problem (COP)*, see also the definition in [AL97] p. 3. For example, *maximizing problem* contains all those instances with $\mathcal{O} = 1$, and *minimizing problem* contains those with $\mathcal{O} = 0$. In this Section, we collect some benchmark problems. These problems can also be found in [KV02], [Law01] and [RK04]. They may involve some definitions in graph theory, readers can refer to Section A. 1 in the Appendix for an explanation of an unknown terminology.

2.2.1 Traveling salesman problem

Traveling salesman problem (TSP) is a classic COP. Formally, TSP concerns finding a cheapest Hamiltonian circuit in a weighted (fully) connected graph $G = (V, E, w)$. Here, recall that a walk W on G is a sequence

$$v_0, (v_0, v_1), v_1, (v_1, v_2), v_2, \ldots, v_n, (v_n, v_{n+1}), v_{n+1}$$

such that each $(v_i, v_{i+1}) \in E$ is an edge (or arc) and each $v_i \in V$ is a vertex, for $i = 0, 1, 2, \ldots, n+1$ and some $n \in \mathbb{N}$. In the sequel, we call the first vertex v_0 on W the *start vertex*, the last vertex v_{n+1} the *end vertex*. And we say that walk W *traverses* a vertex $v \in V$ if $v = v_i$ for some $i = 0, 1, 2, \ldots, n+1$. A Hamiltonian circuit is a particular walk which starts from a vertex, traverses each of other vertex $\in V$ exactly once, and finally returns to the start vertex.

For a walk W in a weighted graph $G = (V, E, w)$, we can define its *traveling cost* $f(W)$ as

$$f(W) := \sum_{e \in E, e \in W} w(e), \tag{2.1}$$

where $e \in W$ indicates that edge (or arc) e is on walk W. Let S be the collection of all Hamiltonian circuits in a weighted (fully) connected graph G, and f defined as (2.1). Then $(S, f, 0)$ is a TSP instance with feasible set S and objective function f. And each Hamiltonian circuit s in the graph G is a feasible solution with the corresponding traveling cost $f(s)$ as its objective value, an optimal solution now is a Hamiltonian circuit with lowest cost. S^* then collects all of the Hamiltonian circuits which have the lowest traveling cost, and the optimum is just the lowest traveling cost.

2.2.2 Assignment problem

Consider that there are a collection $\mathcal{J} = \{J_1, \ldots, J_m\}$ of m jobs and a collection $\mathfrak{M} = \{M_1, \ldots, M_n\}$ of n persons or machines with $m \leq n$. Each job must be processed by exactly one person, and each person can do at most one job. There exists an associated assignment cost c_{ij} if we assign the job J_i to person M_j. Here, an *assignment* is an

injection[1] from $\mathcal{J}$ into $\mathfrak{M}$. Let π be an arbitrary assignment, $\pi(\mathcal{J}_i) \in \mathfrak{M}$ would then represent the assigned machine of job $\mathcal{J}_i$ for each $i = 1, \ldots, m$. The *total (assignment) cost* of π can be defined formally as

$$f(\pi) := \sum_{i=1}^{m} \sum_{j=1}^{n} c_{i,j} \mathbb{1}_{\{\pi(\mathcal{J}_i)\}}(M_j), \tag{2.2}$$

where $\mathbb{1}_{\{\pi(\mathcal{J}_i)\}}(\cdot)$ is the indicator function[2] for the singleton $\{\pi(\mathcal{J}_i)\}$. The assignment problem (AP) concerns finding an assignment with lowest total cost.

Given a set $\mathcal{J}$ of m jobs, a set $\mathfrak{M}$ of n machines with $0 < m \leq n$, and the associated assignment costs matrix $(c_{i,j})_{m \times n}$. And let S denotes the collection of all possible injections (assignments) from $\mathcal{J}$ to $\mathfrak{M}$. Then $(S, f, 0)$ with f defined in (2.2) is an AP instance. Of cause, an optimal solution to this instance is an injection with lowest total cost.

2.2.3 Maximal cut problem

A *cut* to a weighted graph $G = (V, E, w)$ is a partition[3] (V_1, V_2) of the vertices V. Let (V_1, V_2) be a cut, the *cutting benefit* is defined as

$$f\big((V_1, V_2)\big) := \sum_{a \in V_1, b \in V_2} w\big((a, b)\big). \tag{2.3}$$

The maximal cut problem (MaxCut) concerns finding a cut for the underlying weighted graph which has maximal cutting benefit. Obviously, when a weighted graph G is given, and let S be the collection of possible cuts, then $(S, f, 1)$ is a MaxCut instance with objective function f defined in (2.3), and an optimal solution is a cut with maximal cutting benefit.

2.2.4 Knapsack problem

Suppose that there are n items which are tagged as $1, \ldots, n$, and the i-th item is of weight w_i and of value v_i, and there is only one knapsack which has a weight limitation w. The knapsack problem (KP) is a problem which concerns finding a way to pack some of the items into a knapsack such that the total weight of items packed is less than or equivalent to the weight limitation, but the total value is as large as possible.

Given a collection of n items which have values vector $(v_i)_n$ and weights vector $(w_i)_n$, and a knapsack with weights limitation w. Then the corresponding KP instance is $(S, f, 1)$ with each $s \in S$ is a subset of the items collection which has total weights less than or equals to the limitation w, and f is the total value, i.e.

$$f(s) := \sum_{i \in s} v_i \text{ for each } s \in S. \tag{2.4}$$

[1] An injection π from A to B is a map such that for any $a, b \in A$ $a \neq b \Rightarrow h(a) \neq h(b)$.
[2] The indicator function $\mathbb{1}_A(\cdot)$ for a set A is defined as: $\mathbb{1}_A(x) = 1$ if $x \in A$, otherwise $\mathbb{1}_A(x) = 0$.
[3] A partition of a set A is pair (A_1, A_2) such that $A_1 \cup A_2 = A$ and $A_1 \cap A_2 = \emptyset$.

2.3 Heuristic search

CO instances belonging to the same problem may have a similar structure on solutions spaces (feasible sets). Thereby, it is possible to solve them by a unified method. And the method here is finally translated into a search algorithm or procedure. Extensively, an *algorithm* is a list of instructions which can be executed one by one and eventually achieve a specified task. Here, we can consider an algorithm to a problem as a black box which may contain a specified sequence of operations, such that for any input instance of that problem it outputs a feasible (optimal or near-optimal) solution to that instance. In this Section, we shall make a brief introduction to the so-called heuristic search algorithms (namely, heuristics).

2.3.1 Search algorithms in combinatorial optimization

Generally, algorithms for CO are of three types: exact algorithms, approximation algorithms and heuristic search algorithms.

Exact algorithms [Woe03] are those which guarantee to output an optimal solution for all input instances, examples are some brute-force search algorithms [Tra84]. These algorithms often use a 'clever' enumeration strategy to explore the whole feasible set. One disadvantage of these algorithms is that they may require a prohibitive running time in the case of a relatively large problem size.

Approximation algorithms [WS11] are often much more efficient than exact algorithms. They do not guarantee to output an optimal solution. But they can guarantee on qualities of the output solutions. By a sophisticated study on some features of the underlying problem, these algorithms may employ some 'particular' tactics to drive an efficient search in the feasible set, and control the relative errors of the output objective values in a 'low' level. An obstacle here is that they may require much prior knowledge on the structure of the underlying problem or its solutions. Therefore, an approximation algorithm for a particular problem can not apply easily to other problems.

In English, the word 'heuristic' means allowing someone to discover things by his/her self and learning from his/her own experience. Similarly, in optimization, *heuristic search algorithms* refer to these 'intelligent' or 'self-adaptive' search procedures which may start from a purely random search or some fixed solution(s), then automatically accumulate knowledge from their search history (typically some information concealed in good solutions seen) and again use this knowledge to guide their subsequent search. Similar definitions for heuristic search can be found in e.g. [SVVdW80] and [Nic07].

Examples of heuristic search are genetic algorithms [Mic96], simulated annealing [BT93], tabu search [GL99], ant colony optimization [DS04], cross entropy algorithms [RK04], estimation of distribution algorithms [HP11], artificial bee colony [KGOK12], particle swarm optimization [KE95], electromagnetism-like algorithms [BF03] etc. These algorithms are generally rather efficient, but do not guarantee on qualities of the output solutions. However, various applications in the literature showed that they can output high-quality solutions.

Heuristic search algorithms can apply easily to many very tough problems in op-

erations research like vehicle routing, scheduling, tour planning etc. They are rather portable in practice, and are often seen as general-purpose tools for optimization. Sometimes, they are even the unique tool for a practical problem, especially when the problems are very complex.

2.3.2 A classification to heuristic search: solution-based *v.s.* model-based

According to [ZBMD04], heuristic search algorithms can be collected in two categories: solution-based search (SBS) and model-based search (MBS) .

In general, solution-based heuristics may iteratively manipulate present solution(s) hoping that better solution(s) can be found in the next round. Many traditional heuristic algorithms are of solution-based type, e.g. genetic algorithms [Mic96], simulated annealing [BT93], tabu search [GL99] etc. Here, new solutions are often produced by changing and/or exchanging components on present solutions. For example, in a simulated annealing, new solution is often a 'neighbor' of the present solution which is typically produced by changing one or several components (e.g. edges in a tour) on the present solution; in a genetic algorithm, new solutions (children) are generated by performing a crossover and a mutation on the present solutions (parents), where crossover is typically used to exchange components on two mated parents and mutation is typically used to randomly change a component on the resulting solutions after crossover. Therefore, new solution(s) in these traditional algorithms may preserve certain 'good properties' of the present solution(s).

More recent examples of solution-based heuristics are those swarm intelligence algorithms, including artificial bee colony [KGOK12], particle swarm optimization [KE95], electromagnetism-like algorithms [BF03] etc. They aim to simulate the collective or collaborated behavior of a school of practical creatures or physical particles. They employ a swarm of agents, and the agents will do 'parallel' searches in the feasible set. In each iteration, each agent constructs a new solution by manipulating its present solution with reference to its own search history and/or histories of its companions. For example, in a particle swarm optimization algorithm, an agent may construct a new solution with reference to both the best solution seen by itself and the global best solution seen by the whole swarm. As a further reference to SBS, we collect genetic algorithms, simulated annealing and particle swarm optimization in Section A.7 in the Appendix.

This thesis concentrates only on MBS algorithms. Different from SBS, they do not concentrate on particular solutions, but on the solution-production mechanism. This mechanism often explains the structure of the underlying problem to some extent. It is therefore called a 'model' of the problem or its solutions. Typically, the model is a distribution on the feasible set and new solutions are produced by sampling from it. Examples of MBS cover the famous ant colony optimization [DS04], cross entropy algorithms [RK04], estimation of distribution algorithms [HP11] etc. In these algorithms, the models actually reflect an 'empirical' intuition where the optimal solutions are more likely distributed. We take an ant algorithm as an example. Here, the model is a matrix of the 'pheromone' values which serves as the generator for new solutions. Initially, pheromones matrix is generally set to be uniform, if there is no prior information avail-

able. This will result in a purely random search in the first iteration. This reflects an initial intuition that every feasible solution is possible to be an optimal solution before evaluation. After several iterations, the matrix has been empirically updated (typically biased to some best solutions seen), it is now no longer uniform. The resulted search is possibly biased to a particular area on the solutions space. Now, it reflects an experienced intuition that the optimal solutions are more likely distributed in some particular area.

Generally, model-based search will iteratively evolve models. In each iteration, they sample some solutions from the present model. Then, an empirical model will be learned from some selected 'best' solutions in the present sample and/or history. These solutions are typically selected based on their qualities. Hence, the learned model may concentrate on some good solutions. The model for next iteration is eventually constructed with reference to the present model and the learned empirical model. It is therefore biased to good solutions. So, we may expect that in the next sampling, more good solutions will be generated.

A common motivation in MBS is to reach a model which can concentrate only on optimal solutions. To achieve this, they often specify a family of candidate models. And the evolution is generally restricted in that family. Model-based search actually result in a search in the model family. They hope they can reach an 'optimal' model i.e. a model producing optimal solutions only or with an overwhelming probability. Therefore, intrinsically MBS optimize models. This is the most significant difference from SBS.

To further understand MBS, we will collect some popular MBS algorithms in practice in Chapter 3.

3 A short tutorial on model-based search algorithms used in practice

As a background, we have briefly introduced CO (combinatorial optimization) and heuristic search in last Chapter. From now on, we shall concentrate on our main topic: MBS i.e. model-based search (heuristics). This Chapter shall serve as a (short) tutorial for those MBS algorithms used in practice.

MBS algorithms have been applied successfully to many very tough COPs. For example, traveling salesman problem, see [DBKMR05], [Rub99], [DMC96], [DG97b], [OHS$^+$11] etc; maximal cut problem, see [Rub02], [LDM09] etc; vehicle routing problem, see [CHdM05], [BHS99] etc; traffic assignment problem e.g. [ML09]; job shop scheduling, see [SBW11], [CDMT94] etc; buffer allocation e.g. [AKRR05]. According to their originations, MBS algorithms in practice can be collected in three classes: cross entropy algorithms [RK04], ant colony optimization [DS04] and estimation of distribution algorithms [HP11].

Cross entropy algorithm (CE) was initially motivated for rare event simulation, see [Rub97], [Rub99] or A.6 in Appendix. It can be strictly derived by importance sampling under the so-called Kullback-Leibler divergence, see [Rub99]. It has many variants available in the literature, see e.g. [Rub99], [CJK07], [Mar05], [WK14b]. In this Chapter, we will concentrate on the initial version presented in [Rub99]. However, as a reference, we will also make a very short introduction to the additional features in its variants.

Ant colony optimization (ACO) is a large class of algorithms which mimics the foraging behavior of a colony of real ants, see [DS04] for an overview. In this Chapter, we concentrate only on four representative algorithms, namely ant system (AS, see [DMC96]), ant colony system (ACS, see [DG97b]), $\mathcal{MAX}$-$\mathcal{MIN}$ ant system ($\mathcal{MMAS}$, see [SH00]) and population-based ant system (PBAS, see [GM02]). We formally define these four algorithms. As a useful reference, we also describe the foraging behavior of real ants.

Estimation of distribution algorithms (EDA, see [HP11]) is a common name for many algorithms which were initially motivated for efficiently simulating genetic algorithms by probabilistic models. According to the employed models, estimation of distribution algorithms can be collected in two classes: algorithms with *univariate marginal models* and algorithms with *multivariate marginal models*. Here, univariate marginal models are distributions with independent marginals, and multivariate marginal models are those with dependent marginals. Algorithms with univariate marginal models are e.g. univariate marginal distribution algorithm (UMDA, see [MP96]), the population based incremental learning (PBIL, see [Bal94]), and the compact genetic algorithm (CGA, see [HLG99]). Algorithms with multivariate marginal models are e.g. Bayesian optimization algorithm (BOA, see [HGC95], [PM98], [PGCP00], [PGL02], [Pel05]), bivariate marginal distribu-

tion algorithm (BMDA, see [MP96]), mutual information maximizing input clustering (MIMIC, see [DBIV$^+$97]) and extended compact genetic algorithm (ECGA, see [SG00]). Generally, it is difficult to apply an algorithm with multivariate marginal models to optimization problems in practice. The main reason is that learning a multivariate model itself may require a lot of computational effort. Therefore, we will concentrate only on algorithms with univariate marginal models.

The whole Chapter is arranged as: Section 3.1 formalize a common motivation for these algorithms; Section 3.2 introduces the cross entropy algorithms; Section 3.3 introduces the ant colony optimization algorithms; Section 3.4 introduces the estimation of distribution algorithms.

3.1 A common motivation

As mentioned in Section 2.3.2, MBS algorithms do not concentrate on particular solutions, but on models. Here, a *model* is a mechanism which can be used to produce (random) solutions for a CO instance. Generally, it can be expressed as a finite list of parameters, namely a finite vector of reals, due to the finiteness of the underlying feasible set. Typically, it is a distribution[1] on the underlying feasible set, and new solutions are generated by sampling.

MBS algorithms often restrict their models in a specified family. A common motivation here is that they want to find out an 'optimal' model from that family which may produce optimal solutions only (or with an overwhelming probability). To achieve this, they may arbitrarily fix an initial model in that family, and then iteratively evolve the model in the specified family by some fixed strategies. Intrinsically, they optimize models or the solution-production mechanism.

Assume that $(S, f, 0)$ is the underlying CO instance. Let $\mathbb{P}$ be a specified family of models on S satisfying the *basic requirement* that

$$\text{there exists } \mathbf{\Pi}^* \in \mathbb{P}: \quad \mathbf{\Pi}^* \text{ only produces optimal solutions} \tag{3.1}$$

where S^* is the collection of optimal solutions. And we denote the collection of models $\in \mathbb{P}$ which satisfies (3.1) as $\mathbb{P}^*$. Obviously, $\emptyset \neq \mathbb{P}^* \subseteq \mathbb{P}$ under condition (3.1). Then, the common motivation of MBS can be formally stated as an 'optimization' task picking out an 'optimal model' $\mathbf{\Pi}^* \in \mathbb{P}^*$ from candidates $\in \mathbb{P}$.

To do that, these algorithms generally employ a stochastic search on $\mathbb{P}$. Let $\mathbf{\Pi}_0 \in \mathbb{P}$ be an arbitrarily fixed *initial model*. They then iteratively evolve their models through the following three phases:

[1] See A. 5 in the Appendix for a formal definition of distributions.

Crude MBS procedure

Sampling: draw a sample $\mathbf{X}_t$ of a specified size from current model $\mathbf{\Pi}_t \in \mathbb{P}$;

Learning: learn an empirical model $\mathbf{W}_t \in \mathbb{P}$ from some 'best' solutions in $\mathbf{X}_t$ and/or in past iterations, by a specified *learning rule*;

Update: construct a new model $\mathbf{\Pi}_{t+1} \in \mathbb{P}$ with reference to the present model $\mathbf{\Pi}_t$ and the empirical model $\mathbf{W}_t$, by a specified *update rule*.

Here, $\mathbf{W}_t$ is a model concentrating on some 'best' solutions seen currently and/or in the past. With reference to $\mathbf{W}_t$, the next model $\mathbf{\Pi}_{t+1}$ can be biased to 'good solutions'. Thereby, we may expect that more good solutions will be produced in the next round.

Along with MBS algorithms, it is an iteratively *single-start* 'local' search procedure on the specified model family $\mathbb{P}$. Here, 'single-start' means that we start from a single model and construct only one model for the next round (namely $\mathbf{\Pi}_{t+1}$) in each iteration. With word 'local', we mean that $\mathbf{\Pi}_{t+1}$ is actually a 'neighbor' of $\mathbf{\Pi}_t$ since it is constructed also with reference to $\mathbf{\Pi}_t$. It may not be 'far' way from $\mathbf{\Pi}_t$ on the space $\mathbb{P}$. This 'localism' can make the movement of models smoothly on the $\mathbb{P}$. It actually reflects a underlying conservatism. We hope that $\mathbf{\Pi}_{t+1}$ can assimilate the good information concealing in $\mathbf{W}_t$, but we may not hope that $\mathbf{\Pi}_{t+1}$ is completely a copy of $\mathbf{W}_t$. Otherwise, it may be very difficult for the algorithm to escape from the local trap which may be formed by $\mathbf{W}_t$. In the remaining of this Chapter, we will further understand the motivation and this underlying search on $\mathbb{P}$ through some MBS algorithms used in practice.

3.2 Example I: cross entropy algorithm

Cross entropy algorithm (CE) is a very typical MBS algorithm. It was invented by R.Y. Rubinstein, see [Rub99]. Initially, it was motivated for rare-event simulation in complex network simulation, see [Rub97] or A. 6 in the Appendix. Then it was realized as a good and generic optimization tool for both continuous and discrete optimization, see [Rub99]. For a detailed book on CE in CO, see [RK04]. For a concise and helpful tutorial, see [DBKMR05]. There are also several CE variants available in the literature, see e.g. [RK04], [Mar05], [CJK07] and [WK14b]. Here, we stick on the original CE version presented in [Rub99]. As a reference, we shall also make a short introduction to the additional features in its variants.

3.2.1 Representation of feasible solutions

Recall that MBS algorithms optimize models instead of solutions. And the models are restricted in a specified family. In CE, models are actually distributions on the underlying feasible set. To explicitly express the models in CE, we need to formalize the feasible solutions.

Without loss in generality, we assume here a minimizing CO instance $(S, f, 0)$. Note that any maximizing instance can be easily translated into a minimizing instance. For example, let $(S', f', 1)$ be an maximizing instance, then $(S', -f', 0)$[2] is an minimizing instance which has the same feasible set and optimal solutions as $(S', f', 1)$.

In CE, feasible solutions are often assumed to be fixed length strings over a finite alphabet[3], see e.g. [Rub99], [CJK07], [WK14b]. This is reasonable. Actually, for each CO instance, we can represent its feasible solutions in this fashion, due to finiteness of its feasible set. We take an MaxCut (Maximal Cut, see Subsection 2.2.3) instance as an example. Here, a cut (V_1, V_2) on the underlying graph is a partition for the vertices V. Let n denote the size of V. Then, a cut can be uniquely represented as a string $(b_1, \ldots, b_n)$ of length n over the alphabet $\{0, 1\}$, if we employ a rule (encoding) that

$$b_i = \begin{cases} 0 & \text{if the } i\text{-th vertex} \in V_1, \\ 1 & \text{otherwise,} \end{cases}$$

for each $i = 1, \ldots, n$. Therefore, we now think that each feasible solution $s \in S$ of the underlying instance is a string $(a_1, \ldots, a_L)$ where each *item* $a_i \in \mathcal{A}$ for $i = 1, \ldots, L$, $\mathcal{A}$ is a finite *alphabet* and $L \in \mathbb{N}$ is a fixed *solutions length*. Obviously, $S \subseteq \mathcal{A}^L$.

Generally, CE may further assume that $S = \mathcal{A}^L$, see e.g. [DBKMR05] and [CJK07]. Here, we follow this assumption too. Of cause, it may occur that $S \neq \mathcal{A}^L$ in practice. If this is the case, we can assign a *penalty* objective value to those infeasible strings so as to make all strings in $\mathcal{A}^L$ 'feasible'. We can employ an arbitrary value v_{max} as the penalty value where

$$v_{max} > f_{max} = \max\{f(s) \mid s \in S\}.$$

[2]Here, $-f'$ is a function defined as $-f'(s) = -1 \cdot f(s)$ for each $s \in S'$.
[3]An alphabet is a set of items, typically these items are letters.

Then, $(S, f, 0)$ can be extended into an CO instance $(\mathcal{A}^L, f', 0)$ with

$$f'(s) = \begin{cases} f(s) & \text{if } s \in S, \\ v_{max} & \text{otherwise i.e. } s \text{ is infeasible.} \end{cases}$$

We can equivalently solve $(A^L, f', 0)$ instead of $(S, f, 0)$.

3.2.2 The specified model family for cross entropy algorithm

With the above representation i.e. $S = \mathcal{A}^L$ for some finite $\mathcal{A}$ and $L \in \mathbb{N}$, we are now ready to show the commonly used models in CE. Let $\mathbb{P}(\mathcal{A})$ be the class of all possible distributions on the alphabet $\mathcal{A}$ for the underlying instance $(S, f, 0)$. Obviously, each $\pi \in \mathbb{P}(\mathcal{A})$ can be represented as a vector of length $|\mathcal{A}|$ over $\mathbb{R}_+ \cup \{0\}$, i.e.

$$\pi = \big(\pi(a)\big)_{a\in\mathcal{A}} \text{ with each } \pi(a) \geq 0 \text{ and } \sum_{a\in\mathcal{A}} \pi(a) = 1.$$

It describes a mechanism for producing random items or letters from $\mathcal{A}$. We define further that

$$\mathbb{P}_{ce} := \mathbb{P}(\mathcal{A}) \times \mathbb{P}(\mathcal{A}) \times \cdots \times \mathbb{P}(\mathcal{A}) = \mathbb{P}(\mathcal{A})^L. \tag{3.2}$$

Generally, CE algorithms specify $\mathbb{P}_{ce}$ as the model family.

Obviously, each model $\mathbf{\Pi} = \big(\mathbf{\Pi}(1), \ldots, \mathbf{\Pi}(L)\big) \in \mathbb{P}_{ce}$ with each $\mathbf{\Pi}(i) \in \mathbb{P}(\mathcal{A})$ is a product distribution on the product space $\mathcal{A}^L = \mathcal{A} \times \cdots \times \mathcal{A}$. And a random string $s = (a_1, \ldots, a_L) \in \mathcal{A}^L$ is then sampled with a probability

$$\mathbf{\Pi}(s) = \prod_{i=1}^{L} \mathbf{\Pi}(i)\big(a_i\big),$$

where observe that $\mathbf{\Pi}(i) = \Big(\mathbf{\Pi}(i)\big(a\big)\Big)_{a\in\mathcal{A}} \in \mathbb{P}(\mathcal{A})$, for each $i = 1, \ldots, L$, describing a selection in the collection $\mathcal{A}$. Of cause, $\mathbf{\Pi}$ can also be written in details as a vector of reals

$$\mathbf{\Pi} = \big(\mathbf{\Pi}(i)\big)_{i=1,\ldots,L} = \big(\mathbf{\Pi}(a;i)\big)_{a,\in\mathcal{A};i=1,\ldots,L} \text{ with } \mathbf{\Pi}(a;i) := \mathbf{\Pi}(i)\big(a\big).$$

Actually, $\mathbb{P}(\mathcal{A})$ is a closed continuous space i.e. it can be identified as a closed continuous region on the Euclidean Space $\mathbb{R}^n$ where $n = |\mathcal{A}|$. For example, when $|A| = 3$, $\mathbb{P}(\mathcal{A})$ can visually drawn as the gray triangular on Figure 3.1. Since $\mathbb{P}_{ce} = \mathbb{P}(\mathcal{A})^L$ with $L < \infty$, $\mathbb{P}_{ce}$ is then also continuous. Moreover, $\mathbb{P}_{ce}$ is closed under limits i.e. for any convergent sequence $\big(\mathbf{\Pi}_m\big)_{m\in\mathbb{N}}$ with each $\mathbf{\Pi}_m \in \mathbb{P}_{ce}$, we have

$$\lim_{m\to\infty} \mathbf{\Pi}_m \in \mathbb{P}_{ce}.$$

It is not difficult to see that the family $\mathbb{P}_{ce}$ satisfies the basic requirement (3.1) for MBS model family. Let $s^* = (a_1^*, \ldots, a_L^*) \in S^*$ be an optimal solution of the underlying

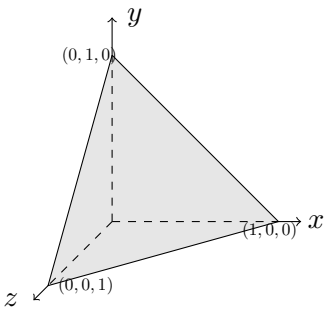


Figure 3.1: A visualization of $\mathbb{P}(\mathcal{A})$ for $|\mathcal{A}| = 3$

instance and $\mathbf{\Pi}^* = \big(\mathbf{\Pi}(1)^*, \ldots, \mathbf{\Pi}(L)^*\big)$ with each $\mathbf{\Pi}(i)^*$ defined

$$\mathbf{\Pi}(i)^*(a) := \begin{cases} 1 & \text{if } a = a_i^*, \\ 0 & \text{otherwise,} \end{cases} \qquad \text{for } i = 1, \ldots, L.$$

Then $\mathbf{\Pi}^*(S^*) = 1$ and $\mathbf{\Pi}^* \in \mathbb{P}_{ce}$. Now, we denote the collection of all $\mathbf{\Pi} \in \mathbb{P}_{ce}$ with $\mathbf{\Pi}(S^*) = 1$ as $\mathbb{P}_{ce}^*$. Evidently, $\emptyset \neq \mathbb{P}_{ce}^* \subseteq \mathbb{P}_{ce}$.

3.2.3 The algorithm

As formulated above, we assume here that the underlying CO instance is $(S, f, 0)$ with $S = \mathcal{A}^L$ where $\mathcal{A}$ is a finite alphabet and L is a fixed solutions length. We take the product distributions family $\mathbb{P}_{ce}$ on $\mathcal{A}^L$ as the specified model family.

CE is a very typical MBS algorithm. It aims to find out an optimal model $\mathbf{\Pi}^* \in \mathbb{P}_{ce}^*$. Starting from an arbitrarily fixed initial model $\mathbf{\Pi}_0 \in \mathbb{P}_{ce}$, it then iteratively improves the present model by shifting it towards its sampled *elite* solutions. It first draws a sample $\mathbf{X}_t$ from the present model $\mathbf{\Pi}_t \in \mathbb{P}_{ce}$, and estimates an empirical model $\mathbf{W}_t \in \mathbb{P}_{ce}$ from a fixed number of elite solutions in $\mathbf{X}_t$. Then, it constructs the model $\mathbf{\Pi}_{t+1} \in \mathbb{P}_{ce}$ for the next round as a convex combination of $\mathbf{\Pi}_t$ and $\mathbf{W}_t$ i.e. in a vector level

$$\mathbf{\Pi}_{t+1} = (1 - \rho)\mathbf{\Pi}_t + \rho \cdot \mathbf{W}_t$$

where $\rho \in (0, 1)$ is a fixed *smooth parameter*. Since $\mathbf{W}_t$ is the empirical distribution of elite solutions, $\mathbf{\Pi}_{t+1}$ will be biased to these solutions. Hence, we may hope that more good solutions can be sampled in the next round. The smooth parameter ρ takes a crucial role. It reflects the relative impact of $\mathbf{W}_t$ in the combination.

Now, we define the CE algorithm presented in [Rub99] in details.The input parameters for CE are:

- a uniform *starting model* $\mathbf{\Pi}_0 \in \mathbb{P}_{ce}$;
- a constant *smooth parameter* $\rho \in (0, 1)$;
- a fixed *sample size* $N \in \mathbb{N}$;
- a small *elite rate* $\alpha \in (0, 1)$ with $\alpha \cdot N \geq 1$.

CE strictly obeys the crude MBS procedure described on p. 15. The three phases here can be defined in details as following.

Algorithm: cross entropy

Sampling: We generate a random sample $\mathbf{X}_t = \left(\mathbf{X}_t^{(1)}, \ldots, \mathbf{X}_t^{(N)}\right)$ from the present model $\mathbf{\Pi}_t = \left(\mathbf{\Pi}_t(i)\right)_{i=1,\ldots,L} \in \mathbb{P}_{ce}$. Here, each $\mathbf{X}_t^{(j)}$ is a random string in $\mathcal{A}^L$ and can be represented in details as

$$\left(\mathbf{X}_t^{(j)}(1), \mathbf{X}_t^{(j)}(2), \ldots, \mathbf{X}^{(j)}(L)\right)$$

where each $\mathbf{X}_t^{(j)}(i)$ is a random item in $\mathcal{A}$ chosen independently by $\mathbf{\Pi}_t(i)$ for $i = 1, \ldots, L$.

Learning: We calculate the objective values of the sampled solutions in $\mathbf{X}_t$, and order them as

$$f(\mathbf{X}_t^{(n_1)}) \leq f(\mathbf{X}_t^{(n_2)}) \leq \cdots \leq f(\mathbf{X}_t^{(n_N)}).$$

We pick out the best $\lfloor \alpha \cdot N \rfloor$ solutions and collect them in $\mathcal{N}_t^b$ i.e.

$$\mathcal{N}_t^b = \left\{\mathbf{X}_t^{(n_1)}, \ldots, \mathbf{X}_t^{(n_{\lfloor \alpha \cdot N \rfloor})}\right\}$$

where $\alpha \in (0, 1)$ is the fixed elite rate. For each item $a \in \mathcal{A}$ and position $i \in \{1, \ldots, L\}$, we calculate a *relative frequency* $\mathbf{W}_t(a; i)$ in the elite solutions $\mathcal{N}_t^b$ as

$$\mathbf{W}_t(a; i) := \frac{\sum_{j=1}^{\lfloor \alpha N \rfloor} \mathbb{1}_{\{a\}}\left(\mathbf{X}_t^{(n_j)}(i)\right)}{\lfloor \alpha N \rfloor}. \tag{3.3}$$

And then, we collect all these frequencies in

$$\mathbf{W}_t := \left(\mathbf{W}_t(1), \ldots, \mathbf{W}_t(L)\right) = \left(\mathbf{W}_t(a; i)\right)_{a \in \mathcal{A}; i=1,\ldots,L} \tag{3.4}$$

with each $\mathbf{W}_t(i) := \left(\mathbf{W}_t(a; i)\right)_{a \in \mathcal{A}}$ for $i = 1, \ldots, L$.

Update We construct the next model $\mathbf{\Pi}_{t+1}$ as a convex combination of $\mathbf{\Pi}_t$ and $\mathbf{W}_t$ i.e.

$$\mathbf{\Pi}_{t+1} = (1 - \rho)\mathbf{\Pi}_t + \rho \mathbf{W}_t \tag{3.5}$$

In details,

$$\mathbf{\Pi}_{t+1}(a; i) = (1 - \rho)\mathbf{\Pi}_t(a; i) + \rho \mathbf{W}_t(a; i) \tag{3.6}$$

for each $a \in \mathcal{A}$ and $i = 1, \ldots, L$.

To make the above algorithm more clearly, a pseudo code is listed in Figure 3.2 on p. 22.

Note that, each $\mathbf{W}_t(i) = \left(\mathbf{W}_t(a; i)\right)_{a \in \mathcal{A}}$ in the above algorithm is actually a distribu-

tion on $\mathcal{A}$, since

$$\sum_{a\in\mathcal{A}} \mathbf{W}_t(a;i) = \sum_{a\in\mathcal{A}} \frac{\sum_{j=1}^{\lfloor \alpha N\rfloor} \mathbb{1}_{\{a\}}\big(\mathbf{X}_t^{(n_j)}(i)\big)}{\lfloor \alpha N\rfloor} = \frac{\sum_{j=1}^{\lfloor \alpha N\rfloor} \sum_{a\in\mathcal{A}} \mathbb{1}_{\{a\}}\big(\mathbf{X}_t^{(n_j)}(i)\big)}{\lfloor \alpha N\rfloor}$$
$$= \sum_{j=1}^{\lfloor \alpha N\rfloor} \frac{1}{\lfloor \alpha N\rfloor} = 1.$$

Thereby,

$$\mathbf{W}_t = \big(\mathbf{W}_t(1),\ldots,\mathbf{W}_t(L)\big) = \big(\mathbf{W}_t(a;i)\big)_{a\in\mathcal{A};i=1,\ldots,L}$$

is a model in $\mathbb{P}_{ce}$. Actually, $\mathbf{W}_t$ maximizes the *likelihood* function of elite solutions $\mathcal{N}_t^b$ i.e. it maximizes

$$\sum_{j=1}^{\lfloor \alpha\cdot N\rfloor} \ln \mathbf{W}(\mathbf{X}_t^{(n_j)}) \quad \text{for } \mathbf{W}\in\mathbb{P}_{ce}. \tag{3.7}$$

Moreover, it is also the model in $\mathbb{P}_{ce}$ which is closest to the truncate distribution

$$\mathbf{W}_t^*(s) = \frac{\mathbb{1}_{\{s'\in S|f(s')\le\gamma_t\}}(s)\mathbf{\Pi}_t(s)}{\sum_{s''\in S}\mathbb{1}_{\{s'\in S|f(s')\le\gamma_t\}}(s'')\mathbf{\Pi}_t(s'')} \quad \text{for each } s\in S \tag{3.8}$$

under the so-called Kullback-Leibler distance (see A. 2 in the Appendix) where $\gamma_t = f\big(\mathbf{X}_t^{(n_{\lfloor\alpha\cdot N\rfloor})}\big)$, see [Rub99] for a proof.

Since $\mathbf{W}_t \in \mathbb{P}_{ce}$ and $\mathbf{\Pi}_t \in \mathbb{P}_{ce}$, $\mathbf{\Pi}_{t+1}$ is also a model in $\mathbb{P}_{ce}$ under update (3.5) and (3.6). And since $\mathbf{W}_t$ approximates the truncate distribution (3.8), $\mathbf{\Pi}_{t+1}$ is biased to the solutions having objective value smaller than γ_t. Thereby, we can hope that more solutions of objective value smaller than γ_t can be sampled in the next round. The smooth parameter ρ reflects the relative importance of the empirical distribution $\mathbf{W}_t$ in the update (3.5). It almost dominates the asymptotic behavior of CE, for details see [WK14b].

The algorithm actually results in a random walk on the continuous space $\mathbb{P}_{ce}$. We start from an initial model (uniform) $\mathbf{\Pi}_0 \in \mathbb{P}_{ce}$. And in each subsequent iteration, we move the present model $\mathbf{\Pi}_t$ to a 'neighbor' $\mathbf{\Pi}_{t+1}$. The move is made smoothly through the update (3.5). The neighbor $\mathbf{\Pi}_{t+1}$ is constructed in a way that it not only preserves some 'local' information in $\mathbf{\Pi}_t$ with a rate $1-\rho$, but also assimilates some 'elite' information (note that $\mathbf{W}_t$ is an approximation of (3.8)) with a rate ρ.

3.2.4 A brief introduction to CE variants

There are also some CE variants, e.g. CE with time-dependent smooth parameters (CE/tdsp, [CJK07]), Ant-like CE (CE/ant, [WK14b]), fully adaptive CE (FACE, [RK04]), graph-based CE with time-dependent smooth parameters (GBCE/tdsp, [Mar05]) and graph-based CE with lower bound on distributions (GBCE/lb, [Mar05]). The essential features of these algorithms are covered in the defined CE. Now we make an introduction to their additional features.

The unique additional feature in CE/tdsp is that it does not employ a fixed smooth parameter, but a sequence of smooth parameters $\left(\rho_t\right)_{t\geq 1}$. And the update (3.5) is therefore changed to

$$\mathbf{\Pi}_{t+1} = (1-\rho_{t+1})\mathbf{\Pi}_t + \rho_{t+1}\mathbf{W}_t$$

where $\mathbf{\Pi}_t$ is the present model and $\mathbf{W}_t$ is calculated by (3.3) and (3.4). Ant-like CE is an extension of CE/tdsp. It introduces a 'feasibility distribution' concept into the above 'Sampling' phase. This concept is inspired by the so-called 'visibility' in ant colony optimization. In the next Chapter, we shall formally introduce this. GBCE/tdsp is a variant of CE/tdsp. It estimates $\mathbf{W}_t$ only from the best solution found in history.

The additional feature in FACE is that it does not fix the sample size. It allows the sample size to vary within a fixed range. In the end of each iteration, it sets the next sample size based on $\mathcal{N}_t^b$. In other word, it employs dynamic sample size.

GBCE/lb is a variant mimicking $\mathcal{MAX}$-$\mathcal{MIN}$ ant system. It introduces a constant lower bound p_{min} to restrict each entry $\mathbf{\Pi}_{t+1}(a;i)$ in $\mathbf{\Pi}_{t+1}$ after the update (3.5) i.e. if $\mathbf{\Pi}_{t+1}(a;i) < p_{min}$, set $\mathbf{\Pi}_{t+1}(a;i) = p_{min}$.

Algorithm Cross entropy algorithm in combinatorial optimization

a) set $t = 0$, initialize $\mathbf{\Pi}_0 \in \mathbb{P}_{ce}$ and design a stop criterion $STOP$;

b) draw a random sample $\mathbf{X}_t = (\mathbf{X}_t^{(1)}, \ldots, \mathbf{X}_t^{(N)})$ by the present distribution $\mathbf{\Pi}_t \in \mathbb{P}_{ce}$ where each $\mathbf{X}_t^{(j)} = \big(\mathbf{X}_t^{(j)}(1), \ldots, \mathbf{X}_t^{(j)}(L)\big) \in \mathcal{A}^L$;

c) evaluate each solution in $\mathbf{X}_t$ according to the objective function f, and order them as

$$f(\mathbf{X}_t^{(n_1)}) \leq \cdots \leq f(\mathbf{X}_t^{(n_N)}),$$

then set $\mathcal{N}_t^b := \{\mathbf{X}_t^{(n_1)}, \ldots, \mathbf{X}_t^{(n_{\lfloor \alpha N \rfloor})}\}$ to be collection of best $\lfloor \alpha N \rfloor$ many solutions;

d) construct an empirical distribution $\mathbf{W}_t \in \mathbb{P}_{ce}$ with each entry

$$\mathbf{W}_t(a; i) = \frac{\sum_{j=1}^{\lfloor \alpha N \rfloor} \mathbb{1}_{\{a\}}\big(\mathbf{X}_t^{(n_j)}(i)\big)}{|\mathcal{N}_t^b|}$$

for $a \in \mathcal{A}$ and $i = 1, \ldots, L$, and then set

$$\mathbf{\Pi}_{t+1} = (1 - \rho)\mathbf{\Pi}_t + \rho\ \mathbf{W}_t;$$

e) while $STOP$ does not hold

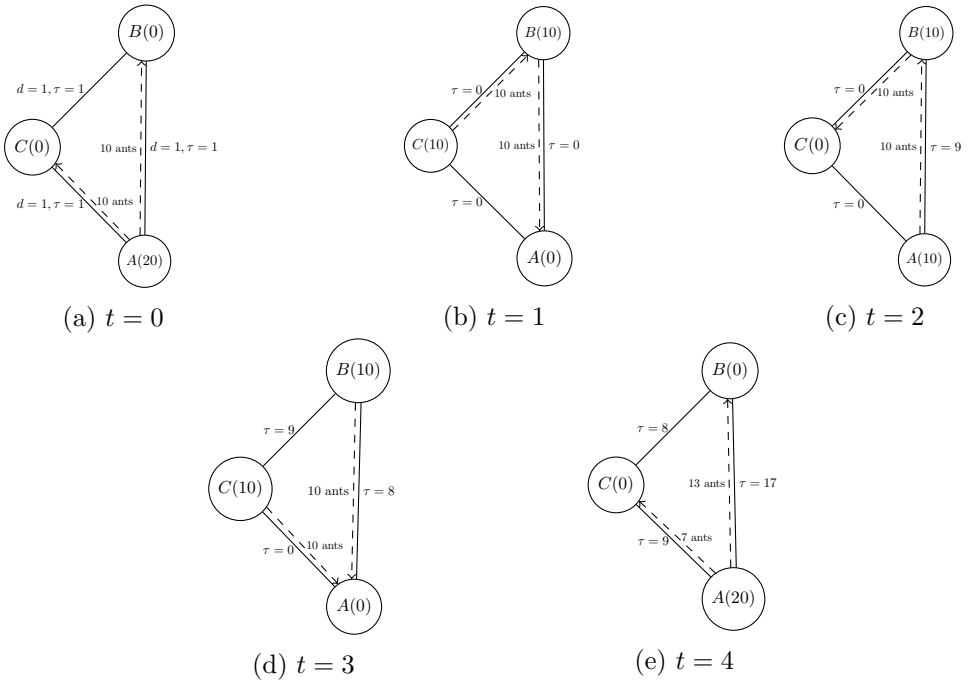
e 1) set $t = t + 1$;

e 2) repeat steps **b)**-**e)**;

Figure 3.2: Cross entropy algorithm in combinatorial optimization

3.3 Example II: ant colony optimization

Ant colony optimization (ACO) is a large class of MBS algorithms which mimics the foraging behavior of a colony of real ants, see [DS04] and [DS10] for an overview. Here, we concentrate on four representative ACO algorithms: ant system, ant colony system, $\mathcal{MAX}$-$\mathcal{MIN}$ ant system and population-based ant system. For a reference of other ACO algorithms, see [DS04].



(a) $t = 0$ (b) $t = 1$ (c) $t = 2$

(d) $t = 3$ (e) $t = 4$

Figure 3.3: A demo of ants foraging behavior

3.3.1 The foraging behavior of ants

ACO algorithms are inspired by the foraging behavior of real ants. Hence, it is helpful to understand this behavior before formally introducing them. Figure 3.3 above clearly demonstrates the foraging behavior.

In practice, ants use a substance called 'pheromone' as a medium to communicate with each other in foraging. When an ant finds a *food source*, it will deposit pheromones on its way back to the *nest*. Other ants can detect the deposited pheromones, and the higher the pheromones on a way the more possible they may follow. The pheromones deposited on a way may also *evaporate* with a certain speed. These can help the ants quickly find and concentrate on a nearest path from their nest to a food source. Figure 3.3 depicts five instants in the foraging procedure for a colony of 20 ants. Here, we assume a nest A, a food source B, and two possible paths from the nest to the unique food source i.e. $A - C - B$ and $A - B$. On the Figure, we use d and τ to describe the

distance and the value of pheromones respectively, e.g. in Subfigure 3.3a we write $d = 1$ and $\tau = 1$ on each edge to mean that the edges are of the same length and the same initial pheromones value. The number inside a bracket following the label of a node represents the number of ants positioned in that node at the corresponding time. For example, in Subfigure 3.3a (t=0) we use $A(20)$ to mean that all ants are positioned in node (nest) A at $t = 0$.

Initially ($t = 0$, Subfigure 3.3a), all ants are positioned in node (nest) A. Since we deposit the same initial pheromones on edges $A - C$ and $A - B$, one half of the ants will follow $A - C$ and others will follow $A - B$ in expectation, see the dashed arrow on Subfigure 3.3a.

At time $t = 1$ (Subfigure 3.3b), 10 ants stand in node C and other 10 ants stand in food source B, where observe that $A - C$ and $A - B$ have the same length, and we assume that all the ants walk with the same speed. Here, we also assume that in 1 unit time, the pheromones on each edge will evaporate 1 unit. So, the pheromones value for each edge at time $t = 1$ equals 0. The ants which has reached food source at $t = 1$ will return back to the nest and deposit pheromones on the path they found i.e $A-B$. Other ants (those in C) will continue to search for a source.

At $t = 2$ (Subfigure 3.3c), 10 ants returned the nest A from the source B, and 10 reached the source B from the node C. Here, we assume that each ant deposits 1 unit of pheromones to each edge on its way back. Thereby, the pheromones value on $A - B$ now is 9 units, where 10 units of pheromones are deposited by the 10 returned ants, but 1 unit of pheromones evaporated. The returned ants (those in the nest A) will still follow path $A - B$ in the next round, because the pheromones value on $A - C$ is 0 (by the evaporation) while the pheromones values on $A - B$ is 9. And those ants arrived in source B will return back and deposit pheromones on the path they found i.e. $A-C-B$.

At $t = 3$ (Subfigure 3.3d), the pheromones value on $C - B$ is 9 i.e. 10 units are deposited by the returned ants and 1 unit evaporated. And now there are again 10 ants in source B and 10 ants in C. And the ants in source B will again go back to nest A along path $A - B$, since this is still the path they found in this round.

At $t = 4$ (Subfigure 3.3e), all the ants returned in nest A, and now the pheromones values on $A - B$ and $A - C$ are 17 units and 9 units respectively. Therefore, we can expect that 7 ants ($\frac{9}{26} \cdot 20 \approx 6.92$) will follow $A-C$ and 13 ants will follow $A-B$ in the next round i.e. the majority of the ants will follow the shorter path $A - B$ when they all completed round trips.

Figure 3.3 actually depicts an *ant cycle* for that colony of 20 ants. Here, an ant cycle means a time interval in which all the ants started simultaneously from their nest and completed at least one round trip between their nest and a food source, and at least one ant finished only one round trip. Since the length of path $A - B$ is 1 unit and the length of path $A-C-B$ is 2 units, the ants following $A-B$ can finish two round trips, but those following $A-C-B$ finish only one round trip in this cycle. As a result, ants deposited totally 20 units of pheromones on $A - B$, but only 10 units on $A - C$ and $C-B$ i.e. the amount of pheromones deposited by ants are proportional to the qualities of the paths (reciprocal of the path length). Due to evaporation, in the end of this cycle the remained pheromones on $A - B$, $A - C$ and $C - B$ are 17, 9, 8 units respectively.

The shorter path (i.e. $A - B$) is therefore overwhelmingly attractive in the next cycle. ACO algorithms would use this behavior as a basic idea in solving practical problems. They see each iteration as an ant cycle, collect the pheromones on edges in a matrix as the 'model' for producing random solutions. Here, a random solution is generated in a way analogous to the construction of a path by an ant in the foraging. And the pheromones matrix is also updated in a similar way as what the ants and evaporation do in the foraging.

3.3.2 Representation of solutions: walks on a construction graph

To apply the foraging behavior, we first need to construct a graph for the underlying CO instance such that each feasible solution can be represented as a walk on that (construction) graph. Here, a graph G (see also A. 1 in the Appendix) is a pair (V, E) where V is a finite set of vertices and $E \subseteq \{(v_1, v_2) \mid v_1 \in V, v_2 \in V\}$. And a walk W on G is a sequence $v_1, (v_1, v_2), v_2, (v_2, v_3), v_3, \ldots, (v_{k-1}, v_k), v_k$ such that

- $v_i \in V$ for each $i = 1, \ldots, k$,
- each pair $(v_i, v_{i+1}) \in E$ for $i = 1, \ldots, k-1$.

Obviously, walk W can be identically represented as a string of vertices $v_1, v_2, \ldots, v_k$. If E satisfies that

$$\forall v_1 \in V \text{ and } v_2 \in V \quad (v_1, v_2) \in E \Rightarrow (v_2, v_1) \in E,$$

we call G as a *undirected* graph and each pair in E as an *edge*. Otherwise, we call each pair in E as an *arc* and the graph as a *directed* graph. Figure 3.4 below shows examples of an undirected graph and a directed graph. Here, two graphs have the same vertices $V = \{v_1, v_2, v_3, v_4\}$, an edge is drawn as a line (Subfigure 3.4a) and an arc is drawn as a line with arrow (Subfigure 3.4b).

(a) undirected graph (b) directed graph

Figure 3.4: Examples of graphs

Formally, we say that a (directed) graph $G = (V, E)$ is a *construction graph* for a CO instance $(S', f', 0)$ (similarly for the maximizing case) under map Φ if (see also Definition 3.1 in [Gut00])

- there is a unique vertex $v_0 \in V$ which is marked as a *starting vertex*;
- let $\mathcal{W}$ be the collection of those walks W satisfying the following conditions
  a) W starts at v_0,
  b) W traverses (contains) each vertex at most once,

c) let e be the end vertex of W, then

$$\forall v \in V: \quad (e, v) \in E \Rightarrow v \text{ has been traversed by W},$$

d) $\Phi : S' \mapsto \mathcal{W}$ is a bijection[4].

Let $G = (V, E)$ be a construction graph for a CO instance $(S', f', 0)$. We shall call each walk on G satisfying *a*)-*c*) as a *legal* walk to that instance. By condition *b*), a legal walk can contain at most $|V| < \infty$ many vertices. And by condition *c*), a legal walk can not be continued without conflicting condition *b*). For example, there are only two legal walks on Subfigure 3.4b (v_1, v_2, v_3, v_4) and (v_1, v_3, v_4) where we identify v_1 as the start vertex. Although $(v_1, v_2), (v_1, v_2, v_3)$ and (v_1, v_2, v_3, v_2) are walks, they conflict either condition *b*) or condition *c*).

In practice, it may be convenient for us to represent a CO instance through a graph. Typical examples are those tour planning problems in Operations Research like TSP. Here, we can arbitrarily fix a starting vertex, and then a legal walk is a walk which starts from the starting vertex and traverses all vertices exactly once. Actually, for an arbitrary CO instance it is also possible for us to construct a graph although this may not be efficient. Let $(S'', f'', 0)$ be an arbitrary instance, we can define a graph $G = (V, E)$ that $V = S'' \bigcup \{\clubsuit\}$ and $E = \{(\clubsuit, s) \mid s \in S''\}$ where $\clubsuit \notin S''$ is a symbol. Then G is a construction graph for $(S'', f'', 0)$ with the unique starting vertex $\clubsuit$.

Now, we introduce some auxiliary definitions and notations related to construction graph. We call a walk on a construction graph G which satisfies conditions *a*)-*b*) as a *partial* legal walk. Let $y = (v_0, v_1, \ldots, v_n)$ be a partial legal walk, then we shall use $V_{\text{Feasible}}(y)$ to notate the collection of all the vertices $v \in V$ making $(v_0, \ldots, v_n, v)$ a new partial legal walk. Of cause, $V_{\text{Feasible}}(y) = \emptyset$ implies y is a (completed) legal walk. And we shall refer to each vertex in $V_{\text{Feasible}}(y)$ as a *feasible continuation* to walk y provided $V_{\text{Feasible}}(y) \neq \emptyset$.

In the sequel of this Section, we assume without loss in generality that the underlying CO instance is an *minimizing* instance with objective function f and feasible set $S = \mathcal{W}$ where $\mathcal{W}$ is the collection of legal walks on a construction graph $G = (V, E)$. And we assume further that v_0 is the unique start vertex on G for all legal walks.

3.3.3 Ant System: the first ACO algorithm

Now, we are ready to formally define ACO algorithms. Here, we start with ant system. Ant system (AS), see [DMC96], is the first ACO algorithm. It completely mimics the ant foraging behavior. Other ACO algorithms can be seen as variants of this algorithm.

The definition of ant system

AS employs a fixed number of artificial ants to iteratively construct random walks on the underlying graph $G = (V, E)$, and the corresponding search mimics completely the foraging behavior. AS identifies v_0 (the start vertex) as the unique nest for those artificial

[4] A bijection $g : A \mapsto B$ is an injection such that for any $y \in B$ there is an $x \in A$ with $y = g(x)$

ants. And an iteration is seen as an ant cycle. The difference here is that each artificial ant may walk on G only once in each cycle. Initially, a fixed amount of pheromones is deposited on each arc $\in E$. And in each cycle (iteration), all the (artificial) ants start from the fixed nest (i.e. v_0) and independently construct (random) legal walks. When an ant finished a partial legal walk, it may choose a feasible continuation according to a probability proportional to the present pheromones. After all ants completed their (legal) walks, the objective values of the walks are calculated, and the present pheromones on each arc $\in E$ evaporate with a fixed rate (i.e. evaporation rate). Finally, each ant deposits an amount of new pheromones on arcs it traversed in this cycle and the amount is based on the qualities of the constructed walk i.e. the better the walk the more pheromones it deposits. Here, the end vertex of a legal walk is thought as a food source and the objective value is seen as the distance from the nest to that source.

Generally, AS takes four input parameters:

- a fixed number $N \in \mathbb{N}$ of artificial ants;
- a fixed *evaporation rate* $\rho \in (0, 1)$;
- a constant *initial pheromones value* $c_0 > 0$ for each arc $\in E$;
- a positive decreasing *assessment* function $g : \mathbb{R} \mapsto \mathbb{R}_+$.

Then each of its iterations can be formally defined as following, where we use $\tau_t(a, b)$ to represent the pheromones value on arc $(a, b) \in E$ in iteration $t \in \mathbb{N}$ and $\tau_0(a, b) = c_0$ for each arc $(a, b) \in E$.

In the below algorithm, legal walks may be of a different length i.e. may contain different number of vertices. Therefore, we use n_j to represent the number of vertices contained in the j-th legal walk in an iteration. Note that, this n_j is a variable, because the j-th legal walks in different iterations may have different numbers of vertices.

Algorithm: ant system

Construction of Walks: The N artificial ants independently construct N random legal walks $\mathbf{X}_t^{(1)}, \ldots, \mathbf{X}_t^{(N)}$. Here, each legal walk $\mathbf{X}_t^{(j)} = \big(v_0, v_1, \ldots, v_{n_j}\big)$ starts from v_0 and is constructed stepwise with each v_{i+1} chosen according to probabilities

$$\mathbf{P}\big[v \mid y\big] = \begin{cases} \frac{\tau_t(v_i,v)}{\sum_{v' \in V_{\text{Feasible}}(y)} \tau_t(v_i,v')} & \text{if } v \in V_{\text{Feasible}}(y), \\ 0 & \text{otherwise,} \end{cases} \tag{3.9}$$

where $y = (v_0, v_1, \ldots, v_i)$ and n_j is number of those vertices on walk $\mathbf{X}_t^{(j)}$ other than v_0, for $i = 0, \ldots, n_j - 1$ and $j = 1, \ldots, N$.

Evaluation of Walks: Calculate the objective value $f\big(\mathbf{X}_t^{(j)}\big)$ as well as the *assessment value* $g\Big(f\big(\mathbf{X}_t^{(j)}\big)\Big)$for each $j = 1, \ldots, N$. Since g is decreasing, the assessment values actually reflect the qualities of the constructed walks.

Evaporation of Pheromones: The present pheromones on each arc evaporate with a rate ρ, i.e. for each arc $(a, b) \in E$

$$\tau_t(a, b) = (1 - \rho)\tau_t(a, b). \tag{3.10}$$

Deposition of Pheromones: Each ant deposits an amount of new pheromones on the traversed arcs and the amount is exactly the assessment value of its constructed walk, thereby the deposited pheromones $\Delta_t(a, b)$ on an arc $(a, b) \in E$ in this cycle (iteration) can be calculated as

$$\Delta_t(a, b) = \sum_{j=1}^{N} \mathbb{1}_{\{W \in \mathcal{W} \mid (a,b) \in W\}}\big(\mathbf{X}_t^{(j)}\big) \cdot g\Big(f\big(\mathbf{X}_t^{(j)}\big)\Big) \tag{3.11}$$

where $\mathcal{W}$ is the collection of all possible legal walks on the underlying graph G, the notation $(a, b) \in W$ means arc (a, b) is on walk W. As a result, pheromones on an arc $(a, b) \in E$ for the next cycle shall be

$$\tau_{t+1}(a, b) = \tau_t(a, b) + \Delta_t(a, b). \tag{3.12}$$

In application, we often incorporate the above two steps 'Evaporation of Pheromones' and 'Deposition of Pheromones' into a single step called 'Update of Pheromones'. And then the next pheromones on each arc $(a, b) \in E$ are calculated by

$$\tau_{t+1}(a, b) = (1 - \rho)\tau_t(a, b) + \Delta_t(a, b) \tag{3.13}$$

where $\tau_t(a, b)$ represents the pheromones on (a, b) in the beginning of iteration t, and $\Delta_t(a, b)$ is defined in (3.11). As a further reference, the pseudo code of this algorithm is listed in Figure 3.5 on p. 33.

An additional feature for some particular problems: the visibility

In practice, some greedy information for stepwisely constructing a random solution may be available. To take advantage of these information, AS may add a 'visibility' concept in the above 'Construction of Walks'.

Now, we employ the famous traveling salesman problem as an example. Assume that the underlying graph $G = (V, E)$ is fully connected i.e. $E = \{(a, b) \mid a \in V, b \in V\}$, and $d : E \mapsto \mathbb{R}_+$ is a distance (weighted) function on G. An optimal solution here is just a shortest Hamiltonian circle on G. Note that each Hamiltonian circle actually corresponds to a legal walk on G if we fix a vertex in V as the unique starting vertex v_0. Then the corresponding objective value for a legal walk $W = (v_0, a_1, \ldots, a_n)$ is

$$f(W) = d(v_0, a_1) + \sum_{i=1}^{n-1} d(a_i, a_{i+1}) + d(a_n, v_0)$$

where we assume further that $V = \{v_0, a_1, \ldots, a_n\}$ for some $n \in \mathbb{N}$. Due to the particular form of this objective function, it is reasonable to continue a partial legal walk with a more close feasible continuation. To cover this idea, AS may assume that each artificial ants can detect not only the pheromones on an arc but also the length of that arc i.e. they can visually compare the distances from their current positions to the feasible continuations by their eyes. Then, an ant can select a feasible continuation according to a mixture of the pheromones and the distances. Formally, the choice probability (3.9) for a feasible continuation therefore becomes

$$\mathbf{P}\big[v \mid y\big] = \begin{cases} \dfrac{\big[\tau_t(v_i,v)\big]^{\alpha}\big[\frac{1}{d(v_i,v)}\big]^{\beta}}{\sum_{v' \in V_{\text{Feasible}}(y)}\big[\tau_t(v_i,v')\big]^{\alpha}\big[\frac{1}{d(v_i,v')}\big]^{\beta}} & \text{if } v \in V_{\text{Feasible}}(y), \\ 0 & \text{otherwise,} \end{cases} \tag{3.14}$$

where $y = (v_0, v_1, \ldots, v_i)$ is partial legal walk with $i < n$ and each $v_i \in V$, the $V_{\text{Feasible}}(y)$ here is actually $V - \{v_0, v_1, \ldots, v_i\}$, $d(v_i, v)$ represents the length of arc (v_i, v), $\alpha > 0$ and $\beta > 0$ are two fixed constants reflecting the relative importance of pheromones information and distances information respectively. We shall refer to (3.14) as the 'visibility case'.

With (3.14) instead of (3.9) in 'Construction of Walks', it may result in a dynamic balance of the empirical information (pheromones) and the prior greedy information (distances). Initially, since pheromones on arcs are of the same value (i.e. $\tau_t(a, b) \equiv c_0$), the distances shall dominate the choice probability (3.14). The algorithm may therefore perform like a randomized greedy search procedure in an early stage. Hereafter, due to the accumulated empirical information, the role of the greedy information in (3.14) may weaken. The underlying search then is guided by a mixture of the empirical and greedy information. In our theoretical study, we shall show that the empirical information may gradually dominate the search in a long term.

Similarly, the visibility concept can be carried over to other problems which have a similar greedy information. Actually, it can be made more extensive so as to capture all problems. However, we shall leave this issue to Section 4.2. In the remaining of this

Section, we stick only to the non-visibility case. Note that, all other ACO algorithms can include the visibility in a similar way.

On the models

Now, we collect all the pheromones $\tau_t(a,b)$ in a matrix

$$\Gamma_t = \Big(\tau_t(a,b)\Big)_{a\in V, b\in V, (a,b)\in E}$$

for each iteration $t \in \mathbb{N}$ in the defined algorithm (see p. 28). Then, Γ_t is the so-called *pheromones matrix* for iteration t. Obviously, Γ_t serves as a generator for random solutions in iteration t, see (3.9) or (3.14). Thereby, they are the models in AS. Now, we define that

$$\mathbb{P}_{aco} = \left\{ \Gamma \;\middle|\; \Gamma = \Big(\tau(a,b)\Big)_{a\in V, b\in V, (a,b)\in E} \quad \tau : E \to \mathbb{R}_+ \text{ is a real function} \right\} \tag{3.15}$$

where E is the collection of all possible arcs on the underlying graph. Of cause, $\mathbb{P}_{aco}$ meets the basic requirement (3.1) for MBS model family i.e. it contains a model which can produce only optimal legal walks. It is actually the model family for AS.

3.3.4 Ant colony system

Ant colony system (ACS), see [DG97b], is an extension of AS. It also covers some features from Q-Learning [GD$^+$95]. Similar to AS, it formulates the feasible solutions of the underlying CO instance as legal walks on a construction graph and mimics the ant foraging behavior. However, it does not directly inherit the choice probability (3.9) or (3.14) for an ant to choose a feasible continuation, but employs a so-called *pseudo random proportional rule* to guide the choice. Moreover, the update of pheromones here is based not only on the present walks (locally), but also on the best walk found so far (globally).

According to a pseudo random proportional rule, an ant may choose a feasible continuation $v \in V$ as

$$v = \begin{cases} \operatorname{argmax}_{a\in V_{Feasible}(y)} \tau_t(v_i, v) & \text{if } u \geq q, \\ b & \text{if } u < q, \end{cases} \tag{3.16}$$

where $y = (v_0, v_1, \ldots, v_i)$ is the partial legal walk constructed, u is a random variate uniformly distributed over $[0,1]$, $q \in (0,1)$ is a constant fixed in advance, and b is a random vertex chosen according to (3.9) or (3.14) in the visibility case. ACS uses this rule to stepwisely construct a legal walk in 'Construction of Walks'. And after an ant completes a legal walk $W = (v_0, v_1, \ldots, v_n)$, the ant will immediately change the pheromones on each arc belonging to W by applying a so-called *local update*, i.e. for each $i = 0, \ldots, n-1$

$$\tau_t(v_i, v_{i+1}) = (1 - \rho_{local})\tau_t(v_i, v_{i+1}) + \rho_{local} \cdot c_1 \tag{3.17}$$

where $\rho_{local} \in (0,1)$ is a fixed *local evaporation rate*, and $c_1 > 0$ is a fixed incremental of pheromones.

After all ants complete their walks, the walks are evaluated and the *best found* walk $\mathbf{X}_t^{BF}$ is updated i.e. $\mathbf{X}_t^{BF}$ is the best solution among all the walks constructed in iterations $0, 1, 2, \ldots, t$. Then, a *global update* is applied according to $\mathbf{X}_t^{BF}$, i.e. for any arc $(a,b) \in E$,

$$\tau_{t+1}(a,b) = (1-\rho_{global})\tau_t(a,b) + \rho_{global}\Delta_t(a,b) \tag{3.18}$$

where ρ_{global} is a fixed *global evaporation rate*, and

$$\Delta_t(a,b) = \begin{cases} g\big(f(\mathbf{X}_t^{BF})\big) & \text{if } (a,b) \in \mathbf{X}_t^{BF}, \\ 0 & \text{otherwise.} \end{cases} \tag{3.19}$$

with g is a fixed assessment function. See Figure 3.6 on p. 34 for a pseudo code of ACS.

3.3.5 $\mathcal{MAX}$-$\mathcal{MIN}$ ant system

$\mathcal{MAX}$-$\mathcal{MIN}$ ant system ($\mathcal{MMAS}$), see [SH00], is a variant of AS. Similar to AS, it also employs the choice probability (3.9) or (3.14) (for the visibility case) for an ant to choose a feasible continuation. However, it employs a fixed upper bound $\tau_{\max} > 0$ as well as a fixed lower bound $\tau_{\min} > 0$ to restrict pheromones in each iteration. Moreover, new pheromones are deposited only on those arcs which belongs to the best walk found or iteration best walk.

In more details, we assume that all ants have completed their walks according to choice probability (3.9) or (3.14) in a iteration t i.e. the step 'Construction of Walks' on page 28. Then all constructed walks are evaluated, and the best found walk $\mathbf{X}_t^{BF}$ as well as the *iteration best* walk $\mathbf{X}_t^{IT}$ are updated i.e. $\mathbf{X}_t^{IT}$ is the best walk among the constructed walks in the present iteration. To update the pheromones, $\mathcal{MMAS}$ randomly chooses a solution $\mathbf{X}_t^{upd}$ from $\{\mathbf{X}_t^{IT}, \mathbf{X}_t^{BF}\}$, and sets $\tau_{t+1}(a,b)$ for all $(a,b) \in E$ as

$$\tau_{t+1}(a,b) = \begin{cases} \max\big\{(1-\rho)\tau_t(a,b), \tau_{\min}\big\} & \text{if } (a,b) \notin \mathbf{X}_t^{upd}, \\ \min\big\{(1-\rho)\tau_t(a,b) + \rho \cdot g\big(f(\mathbf{X}_t^{upd})\big), \tau_{\max}\big\} & \text{otherwise.} \end{cases} \tag{3.20}$$

where $\rho \in (0,1)$ is a fixed evaporation rate, g is a fixed assessment function, $\tau_{min} > 0$ and $\tau_{max} > 0$ are fixed pheromones lower bound and upper bound. See Figure 3.7 on p. 35 for a pseudo code of $\mathcal{MMAS}$.

3.3.6 Population-based ant system

Population-based ant system (PBAS), see [GM02], is a further variant of $\mathcal{MMAS}$. It constructs random solutions (i.e. legal walks) by the same approach as AS and $\mathcal{MMAS}$. And it covers the main feature of $\mathcal{MMAS}$, i,e. it also restricts the pheromones. But it uses a different (pheromones) update mechanism. It does not use evaporation, and only rewards the solutions which are stored in an *archive*. The archive is designed to store iteration best solutions for some past iterations. Initially, the archive is empty. Then in

each iteration, the present iteration best solution is added to this archive, and an amount of pheromones are deposited on each arc belonging to this solution. When the archive reaches a specified size, one solution in the archive shall be kicked out by a specified *out rule*, and the pheromones which are deposited by this solution are simultaneously removed.

Now, we introduce PBAS in details. Assume that the archive is of size $K \in \mathbb{N}$. In each iteration $t < K$, it first constructs a fixed number of random legal walks according to the step 'Construction of Walks' (see the defined AS on p. 28). Then the present iteration best walk $\mathbf{X}_t^{IT}$ is added to the archive, and the pheromones $\tau_{t+1}(a,b)$ on an arc $(a,b) \in E$ for next iteration is calculated as

$$\tau_{t+1}(a,b) = \begin{cases} \tau_t(a,b) + \frac{\tau_{max}-c_0}{K} g\big(f(\mathbf{X}_t^{IT})\big) & \text{if } (a,b) \in \mathbf{X}_t^{IT}, \\ \tau_t(a,b) & \text{otherwise,} \end{cases} \tag{3.21}$$

where $c_0 > 0$ is the initial pheromones value on each arc, $\tau_{max} > 0$ is a fixed pheromones upper bound, g is the assessment function, and $\mathbf{X}_0^{IT}, \ldots, \mathbf{X}_t^{IT}$ are the iteration best solutions stored in the present archive.

In each subsequent iteration (i.e $t \geq K$), after all random solutions are constructed, a solution which is determined by an *out rule* is removed from the present archive. And the pheromones added according to this solution in previous are simultaneously removed. Let $\mathbf{X}^{out}$ represent the removed solution, then the pheromones are changed as

$$\tau_t(a,b) = \begin{cases} \tau_t(a,b) - \frac{\tau_{max}-c_0}{K} g\big(f(\mathbf{X}^{out})\big) & \text{if } (a,b) \in \mathbf{X}^{out}, \\ \tau_t(a,b) & \text{otherwise.} \end{cases} \tag{3.22}$$

Then, the iteration best solution $\mathbf{X}_t^{IF}$ in present iteration is added to the archive., and pheromones $\tau_{t+1}(a,b)$ on an arc $(a,b) \in E$ for next iteration is again calculated as (3.21).

Generally, there are two out rules: first in first out (FIFO) and worst out (WO). According to FIFO, the oldest solution in the archive is removed. And with WO, we remove the worst solution from the archive. Of cause, it is also possible to employ a random out rule which randomly removes a solution from the archive based on solutions qualities.

With update (3.21) and (3.22), the pheromones $\tau_t(a,b)$ for a iteration $t \geq K$ actually equals

$$\tau_t(a,b) = c_0 + \frac{\tau_{max} - c_0}{K} \sum_{j=1}^{K} g\big(f(\mathbf{x}^{(j)})\big) \mathbb{1}_{\{(c,d) \mid (c,d) \in \mathbf{x}^{(j)}\}}\big((a,b)\big) \tag{3.23}$$

where $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(K)}$ are the iteration best solutions stored in the archive in the beginning of iteration t. Generally, the assessment function g in PBAS is often restricted to have a range $[0,1]$. Thereby, the initial pheromones value c_0 may serve as a pheromones lower bound and τ_{max} may serve as an upper bound. So, PBAS actually covers the essential features of $\mathcal{MMAS}$. As a further reference, we list the pseudo code for PBAS in Figure 3.8 on p. 36.

Algorithm Ant system[a]

a) set $t = 0$, $\tau_0(a, b) = c_0$ for each arc $(a, b) \in E$, select an $N \in \mathbb{N}$, a $\rho \in (0, 1)$ and a stop criterion $STOP$;

b) for $j = 1$ to N;

b 1) $y = (v_0)$;

b 2) while $V_{\text{Feasible}}(y) \neq \emptyset$;

- choose a feasible continuation $v \in V$ by

$$\mathbf{P}(v|y) = \begin{cases} \frac{\tau_t(n_y, v)}{\sum_{v' \in V_{\text{Feasible}}(y)} \tau_t(n_y, v')} & \text{if } v \in V_{\text{Feasible}}(y), \\ 0 & \text{otherwise,} \end{cases}$$

  where n_y is the end vertex of y;
- $y = (y, v)$;
- update $V_{\text{Feasible}}(y)$;

b 3) $\mathbf{X}_t^{(j)} = y$;

c) construct Γ_{t+1} as, for all $(a, b) \in E$

$$\tau_{t+1}(a, b) = (1 - \rho)\tau_t(a, b) + \Delta_t(a, b)$$

and

$$\Delta_t(a, b) = \sum_{j=1}^{N} \mathbb{1}_{\{s \in S | (a,b) \in s\}}(\mathbf{X}_t^{(j)}) g(f(\mathbf{X}_t^{(j)}));$$

d) while $STOP$ does not hold

d 1) $t = t + 1$;

d 2) repeat steps **b)**-**d)**;

[a] We only present the non-visibility case here, the visibility case is similar.

Figure 3.5: Ant system

Algorithm Ant colony system

a) set $t = 0$, initialize Γ_0 and a stop criterion $STOP$;

b) for $j = 1$ to N;

b 1) $y = (v_0)$;

b 2) while $V_{\text{Feasible}}(y) \neq \emptyset$;

- choose a feasible continuation $v \in V$ by

$$v = \begin{cases} \text{argmax}_{v \in V_{\text{Feasible}}(y)} \tau_t(n_y, v) & \text{if } u \geq q, \\ b & \text{if } u < q, \end{cases}$$

  where n_y is the end vertex of y, and b is chosen as in AS;
- $y = (y, v)$;
- update $V_{\text{Feasible}}(y)$;

b 3) apply the local update

$$\tau_t(a,b) = (1 - \rho_{local})\tau_t(a,b) + \rho_{local} \cdot c_1$$

for each arc $(a, b) \in y$;

b 4) $\mathbf{X}_t^{(j)} = y$;

c) update the best solution $\mathbf{X}_t^{BF}$ found so far and construct Γ_{t+1} as

$$\tau_{t+1}(a,b) = (1 - \rho_{global})\tau_t(a,b) + \rho_{global}\Delta_t(a,b)$$

where

$$\Delta_t(a,b) = \begin{cases} g(f(\mathbf{X}_t^{BF})) & \text{if } (a,b) \in \mathbf{X}_t^{BF}, \\ 0 & \text{otherwise,} \end{cases}$$

for each arc $(a, b) \in E$;

d) while $STOP$ does not hold

d 1) $t = t + 1$;

d 2) repeat steps **b)-d)**;

Figure 3.6: Ant colony system algorithm

Algorithm $\mathcal{MAX}$-$\mathcal{MIN}$ ant system

a) set $t = 0$, initialize Γ_0 and select a stop criterion $STOP$;

b) for $j = 1$ to N;

b 1) $y = (v_0)$;

b 2) while $V_{\text{Feasible}}(y) \neq \emptyset$;

- choose a feasible continuation $v \in V$ as AS;
- $y = (y, v)$;
- update $V_{\text{Feasible}}(y)$;

b 3) $\mathbf{X}_t^{(j)} = y$;

c) calculate $\mathbf{X}_t^{IT}, \mathbf{X}_t^{BF}$ and select $\mathbf{X}_t^{upd}$ from $\{\mathbf{X}_t^{IT}, \mathbf{X}_t^{BF}\}$, then construct Γ_{t+1} with $\mathbf{X}_t^{upd}$ as

$$\tau_{t+1}(a,b) = \begin{cases} \max\{(1-\rho)\tau_t(a,b), \tau_{\min}\} & \text{if } (a,b) \notin \mathbf{X}_t^{upd}, \\ \min\{(1-\rho)\tau_t(a,b) + \rho \cdot g(f(\mathbf{X}_t^{upd})), \tau_{\max}\} & \text{otherwise} \end{cases}$$

for each arc (a, b) in E;

d) while $STOP$ does not hold

d 1) $t = t + 1$;

d 2) repeat steps **b)-d)**;

Figure 3.7: $\mathcal{MAX}$-$\mathcal{MIN}$ ant system

Algorithm Population based ant system

a) set $t = 0$, initialize Γ_0 and select a stop criterion $STOP$;

b) for $j = 1$ to N;

b 1) $y = (v_0)$;

b 2) while $V_{\text{Feasible}}(y) \neq \emptyset$;

- choose a feasible continuation $v \in V$ as AS;
- $y = (y, v)$;
- update $V_{\text{Feasible}}(y)$;

b 3) $\mathbf{X}_t^{(j)} = y$;

c) pick out the iteration best solution $\mathbf{X}_t^{IT}$ from $\mathbf{X}_t^{(1)}, \ldots, \mathbf{X}_t^{(N)}$;

d) if $t \leq K$, add $\mathbf{X}_t^{IT}$ to the archive, and construct Γ_{t+1} as

$$\tau_{t+1}(a,b) = \begin{cases} \tau_t(a,b) + \frac{\tau_{max} - c_0}{K} g\big(f(\mathbf{X}_t^{IT})\big) & \text{if } (a,b) \in \mathbf{X}_t^{IT}, \\ \tau_t(a,b) & \text{otherwise,} \end{cases}$$

for each arc in E;

e) if $t \geq K$, use the out rule to remove a solution from the archive, say $\mathbf{X}^{out}$, and

$$\tau_t(a,b) = \begin{cases} \tau_t(a,b) - \frac{\tau_{max} - c_0}{K} g\big(f(\mathbf{X}^{out})\big) & \text{if } (a,b) \in \mathbf{X}^{out}, \\ \tau_t(a,b) & \text{otherwise.} \end{cases}$$

for each $(a,b) \in E$, then add $\mathbf{X}_t^{IF}$ into the archive and construct Γ_{t+1} as step d);

f) while $STOP$ does not hold

f 1) $t = t + 1$;

f 2) repeat steps **b)**-**f)**;

Figure 3.8: Population based ant system

3.4 Example III: estimation of distribution algorithms

Estimation of distribution algorithms (EDA) are initially motivated for efficiently simulating the robust genetic algorithm through probabilistic models, see [PGL02]. For a detailed survey on EDA, see [HP11]. Here, we concentrate only on some practical EDA which employ the so-called univariate marginal models, e.g. univariate marginal distribution algorithm [MP96], the population based incremental learning [Bal94], and the compact genetic algorithm [HLG99]. For some theoretical EDA which concern multivariate marginal models, see the literature [HGC95], [PM98], [PGCP00], [PGL02], [Pel05], [MP96], [DBIV$^+$97] and [SG00] as a reference.

3.4.1 Feasible solutions and univariate models

It is rather popular in genetic algorithms [Mic96] that the feasible solutions of CO instances are represented as binary strings of a fixed length. As EDA are motivated for simulating genetic algorithms, they shall follow this representation. Hence, we assume here that the underlying feasible set $S \subseteq \{0,1\}^L$ for some $L \in \mathbb{N}$. Actually, we can assume further that $S = \{0,1\}^L$ by introducing a penalty objective value for those infeasible binary strings as discussed in CE. Thereby, we consider here, without loss in generality, a minimizing CO instance $(S, f, 0)$ with $S = \{0,1\}^L$ for some $L \in \mathbb{N}$.

Here, we discuss only those EDA which evolve univariate marginal models. Generally, a *univariate marginal model* on $\{0,1\}^L$ is a product distribution $\mathbf{\Pi} = \big(\mathbf{\Pi}(1), \ldots, \mathbf{\Pi}(L)\big)$ on $\{0,1\}^L$ such that each $\mathbf{\Pi}(i) = \big(\mathbf{\Pi}(0;i), \mathbf{\Pi}(1;i)\big)$ is a Bernoulli distribution[5] with success probability $\mathbf{\Pi}(1;i)$ and failure probability $\mathbf{\Pi}(0;i) = 1 - \mathbf{\Pi}(1;i)$ for all $i = 1, \ldots, L$. And by distribution $\mathbf{\Pi}$, a random binary string $s = (b_1, \ldots, b_L) \in \{0,1\}^L$ is then sampled with a probability

$$\mathbf{\Pi}(s) = \prod_{i=1}^{L} \big[\mathbf{\Pi}(0;i)\big]^{\mathbb{1}_{\{0\}}(b_i)} \cdot \big[\mathbf{\Pi}(1;i)\big]^{\mathbb{1}_{\{1\}}(b_i)}$$

where we use a convention that $0^0 = 1$. In details, we can generate a random string $s = (b_1, b_2, \ldots, b_L)$ sequentially by $\mathbf{\Pi}$ with each b_i generated *independently* by the Bernoulli distribution $\mathbf{\Pi}(i)$ for all $i = 1, \ldots, L$. In the sequel, we shall write the family of univariate marginal models on $\{0,1\}^L$ as $\mathbb{P}_{um}$.

Extensively, a univariate marginal model is a mechanism which produces a solution with each of the *building blocks* generated independently. Here, a building block means a component on a solution. For example, an arc is a building block of a walk; a letter is a building block of a string. Obviously, the distributions $\in \mathbb{P}_{ce}$ serve as univariate marginal models in CE. However, the pheromones matrices $\in \mathbb{P}_{aco}$ do not serve as univariate marginal models in ACO, since the selection of a continuing arc for a partial legal walk is dependent on the end vertex of the present partial walk.

[5] The Bernoulli distribution is distribution of a random variable which takes value 1 with a success probability $p \in [0,1]$ and value 0 with probability $1 - p$.

Evidently, we can simplify a univariate model $\mathbf{\Pi} = \big(\mathbf{\Pi}(b;i)\big)_{b\in\{0,1\};i=1....,L} \in \mathbb{P}_{um}$ as a vector $\big(\pi_1,\ldots,\pi_L\big)$, where each $\pi_i := \mathbf{\Pi}(1;i)$ represents the success probability in the i-th Bernoulli distribution $\mathbf{\Pi}(i)$ for $i = 1,\ldots,L$. Then, the corresponding probability of generating a random string $s = (b_1,\ldots,b_L) \in \{0,1\}^L$ can be written as

$$\mathbf{\Pi}(s) = \prod_{i=1}^{L} \big[1-\pi_i\big]^{\mathbb{1}_{\{0\}}(b_i)} \cdot \pi_i^{\mathbb{1}_{\{1\}}(b_i)}.$$

Of cause, there are also some EDA which employs multivariate marginal models. Examples are Bayesian optimization algorithm (BOA, [Pel05]), extended compact genetic algorithm (ECGA, [SG00]), mutual-information-maximizing-input-clustering algorithm (MIMIC, [DBIV$^+$97]), bivariate marginal distribution algorithm (BMDA, [PM98]) and factorized distribution algorithm (FDA, [MMR99]). Generally, a *multivariate marginal model* is a model which concerns mutual interactions of building blocks on some positions in the generation of a random solution. A typical example of multivariate models is the so-called Bayesian network, which describes the mutual dependencies of different positions on a random solution through a directed graph with each node as a position and each arc indicating a probabilistic dependency, see [PM98] for details. Although there are some EDA which employ multivariate marginal models, they can not apply easily to practical optimization problems. The main reason here is that learning a multivariate marginal model (from the sampled data) itself is generally rather time consuming, for example learning a good Bayesian network for a large data may require a prohibitive running time. Hence, we concentrate only on those EDA with univariate marginal models.

3.4.2 Simulating uniform crossover by a univariate marginal model

EDA are initially motivated for simulating genetic algorithms through probabilistic models. Typically, they simulate the uniform crossover. As a reference, we now give a brief introduction to this.

Crossover is a very important operator for a genetic algorithm [Mic96]. It takes a crucial role in the performance of a genetic algorithm. Generally, it takes in two present solutions (parents), and then produce two new solutions (children) hoping that the new solutions can preserve good properties of their parents. For a reference to crossover, see Section A.7 in the Appendix.

Uniform crossover is one of the most frequently used crossover operators in genetic algorithms. Suppose that $p^{(1)} = (b_1,\ldots,b_L) \in S$ and $p^{(2)} = (a_1,\ldots,a_L) \in S$ are two arbitrarily fixed binary strings, where recall that the underlying feasible set S is assumed to be $\{0,1\}^L$. Then, uniform crossover may produce two children $c^{(1)} = (c_1,\ldots,c_L) \in S$ and $c^{(2)} = (d_1,\ldots,d_L)$ for these two parents $p^{(1)}$ and $p^{(2)}$ by setting each

$$c_i = \begin{cases} a_i & \text{if } p_i < 0.5 \\ b_i & \text{if } p_i \geq 0.5 \end{cases} \quad \text{and} \quad d_i = \begin{cases} b_i & \text{if } c_i = a_i \\ a_i & \text{if } c_i = b_i \end{cases} \tag{3.24}$$

where p_i is a random variate generated by the uniform distribution on $[0,1]$, for $i = 1,\ldots,L$.

Let π_i be the frequency of bit 1 in $\{a_i, b_i\}$ i.e.

$$\pi_i = \frac{\mathbb{1}_{\{1\}}(a_i) + \mathbb{1}_{\{1\}}(b_i)}{2}$$

is the relative frequency of bit 1 at position i in the two parents, for each $i = 1, \ldots, L$. Obviously, each c_i in child $c^{(1)}$ can only take values in $\{0, 1\}$, since $a_i, b_i \in \{0, 1\}$. By the uniform crossover (3.24)

$$\begin{aligned}\mathbf{P}\big[c_i =1\big] &= \mathbf{P}\big[c_i = 1, c_i = a_i\big] + \mathbf{P}\big[c_i = 1, c_i = b_i\big] \\ &= \mathbf{P}\big[a_i = 1, c_i = a_i\big] + \mathbf{P}\big[b_i = 1, c_i = b_i\big] \\ &= \mathbf{P}\big[c_i = a_i \mid a_i = 1\big] \cdot \mathbf{P}\big[a_i = 1\big] + \mathbf{P}\big[c_i = b_i \mid b_i = 1\big] \cdot \mathbf{P}\big[b_i = 1\big].\end{aligned}$$

Note that in (3.24), the random event $[c_i = a_i]$ is independent of $[a_1 = 1]$ and the random event $[c_i = b_i]$ is also independent of $[b_i = 1]$. Thereby,

$$\begin{aligned}\mathbf{P}\big[c_i = 1\big] &= \mathbf{P}\big[c_i = a_i\big] \cdot \mathbf{P}\big[a_i = 1\big] + \mathbf{P}\big[c_i = b_i\big] \cdot \mathbf{P}\big[b_i = 1\big] \\ &= 0.5 \cdot \mathbf{P}\big[a_i = 1\big] + 0.5 \cdot \mathbf{P}\big[b_i = 1\big] \\ &= 0.5 \cdot \Big(\mathbf{P}\big[a_i = 1\big] + \mathbf{P}\big[b_i = 1\big]\Big).\end{aligned}$$

Since the two parents $p^{(1)}$, $p^{(2)}$ are fixed, b_i, a_i are also fixed. Therefore,

$$\mathbf{P}\big[a_i = 1\big] = \mathbb{1}_{\{1\}}(a_i) \quad \text{and} \quad \mathbf{P}\big[b_i = 1\big] = \mathbb{1}_{\{1\}}(b_i).$$

As a result,

$$\begin{aligned}\mathbf{P}\big[c_i = 1\big] &= \frac{\mathbf{P}\big[a_i = 1\big] + \mathbf{P}\big[b_i = 1\big]}{2} \\ &= \frac{\mathbb{1}_{\{1\}}(a_i) + \mathbb{1}_{\{1\}}(b_i)}{2} = \pi_i.\end{aligned}$$

This means that each c_i in child $c^{(1)}$ can be equivalently generated by a Bernoulli distribution with success probability π_i. A similar discussion applies to each position d_i on child $c^{(2)}$, although $c^{(2)}$ is completely determined by $c^{(1)}$.

Let $\mathbf{\Pi}_{p^{(1)},p^{(2)}} = \big(\pi_1, \pi_2, \ldots, \pi_L\big)$. Obviously, $\mathbf{\Pi}_{p^{(1)},p^{(2)}} \in \mathbb{P}_{um}$ is a univariate marginal model. Producing a new solution from $p^{(1)}, p^{(2)}$ by uniform crossover (3.24) is then equivalent to drawing a random solution from model $\mathbf{\Pi}_{p^{(1)},p^{(2)}}$. Therefore, we may simulate uniform crossover by sampling from this univariate marginal model.

Actually, we can further think $\mathbf{\Pi}_{p^{(1)},p^{(2)}}$ as a univariate marginal model learned from the 'sample' $\{p^{(1)}, p^{(2)}\}$. Then, uniform crossover in genetic algorithms may serve as a 'Learning-Reproducing' procedure i.e. we learn a model from two parents and then reproduce two children by the learned model. The EDA discussing in this Section are just inspired by this procedure. They may extend this procedure in several directions.

3.4.3 Univariate marginal distribution algorithm

Univariate marginal distribution algorithm (UMDA), [MP96], extends uniform crossover (3.24) for two parents into a Learning-Reproducing procedure for 'multi-parents'. It wants to simulate 'multi-parents' crossover by univariate marginal models.

Let $P = \{p^{(1)}, \ldots, p^{(n)}\}$ be a population of parents, where $n \geq 1$ and

$$\begin{aligned} p^{(1)} &= \big(b_{1,1}, b_{2,1}, \ldots, b_{i,1} \ldots, b_{L,1}\big) \\ p^{(2)} &= \big(b_{1,2}, b_{2,2}, \ldots, b_{i,2} \ldots, b_{L,2}\big) \\ &\cdots\cdots \\ p^{(j)} &= \big(b_{1,j}, b_{2,j}, \ldots, b_{i,j} \ldots, b_{L,j}\big) \\ &\cdots\cdots \\ p^{(n)} &= \big(b_{1,n}, b_{2,n}, \ldots, b_{i,n} \ldots, b_{L,n}\big) \end{aligned}$$

with $b_{i,j} \in \{0, 1\}$ for all $i = 1, \ldots, L$ and $j = 1, \ldots, n$. In a multi-parents uniform crossover, a child $c = (c_1, \ldots, c_L) \in S$ for these n parents can be generated as

$$c_i = b_{i,r_i}, \quad r_i \text{ is a number randomly chosen from } \{1, \ldots, n\} \tag{3.25}$$

for all $i = 1, \ldots, L$, namely each c_i is a random variable uniformly distributed over $\{b_{i,1}, b_{i,2}, \ldots, b_{i,j}, \ldots, b_{i,n}\}$. Obviously, (3.25) extends the uniform crossover (3.24). To simulate (3.25), we can learn a univariate marginal model $\mathbf{\Pi}_P = (\pi_1, \ldots, \pi_L) \in \mathbb{P}_{um}$ such that each π_i is the relative frequency of bit 1 at position i in the population P i.e.

$$\pi_i = \frac{\sum_{j=1}^{n} \mathbb{1}_{\{1\}}(b_{i,j})}{n} \tag{3.26}$$

for $i = 1, \ldots, L$. Then, the child c defined in (3.25) for the parents population P is intrinsically a random solution satisfying distribution $\mathbf{\Pi}_P$.

By (3.26), UMDA can extend the Learning-Reproducing procedure underlying the uniform crossover for two parents to a procedure involving multi-parents. Initially, it samples a number of random solutions by the uniform distribution $\in \mathbb{P}_{um}$, and collect them in a *memory*. In each subsequent iteration, UMDA shall *select* out some promising solutions from the present memory as a parents population P, by a fixed *choice rule*. Then, a univariate marginal model $\mathbf{\Pi}_P \in \mathbb{P}_{um}$ is learned from P by (3.26) and some children for P are produced by sampling from $\mathbf{\Pi}_P$. Finally, the memory for the next round is updated according to a *memory update rule*.

Formally, UMDA may take in the following as input parameters:

- a fixed *memory size* $M \in \mathbb{N}$,
- a fixed *parents population size* $n \in \mathbb{N}$ which is often determined by a fixed *selection rate* $\alpha \in (0, 1)$ i.e. $n = \alpha \cdot M$,
- a fixed *number of children* $N \in \mathbb{N}$,
- a fixed *choice rule* for selecting parents,
- a fixed *memory update rule* for maintaining the memory,

- a *initial memory* $\mathbf{M}_0$ which contains M feasible solutions.

Each iteration $t \geq 0$ of the algorithm can be defined as following. Where $\mathbf{M}_t$ means memory for iteration t.

Algorithm: univariate marginal distribution algorithm

Selection: Select n parents $p_t^{(1)}, \ldots, p_t^{(n)}$ from the present memory $\mathbf{M}_t$ according to the fixed choice rule, and collect them in P_t. Where each $p_t^{(j)} = \left(b_{1,j}^t, \ldots, b_{L,j}^t\right) \in \mathbf{M}_t$ with $b_{i,j}^t \in \{0,1\}$, for $i = 1, \ldots, L$ and $j = 1, \ldots, n$.

Learning: Learn a model $\mathbf{\Pi}_t = \left(\pi_{t,1}, \ldots, \pi_{t,L}\right) \in \mathbb{P}_{um}$ from P_t by rule (3.26), i.e.

$$\pi_{t,i} = \frac{\sum_{j=1}^{n} \mathbb{1}_{\{1\}}\left(b_{i,j}^t\right)}{n}$$

for each $i = 1, \ldots, L$.

Reproduction: Draw a sample (children) $\mathbf{X}_t = \left(\mathbf{X}_t^{(1)}, \ldots, \mathbf{X}_t^{(N)}\right)$ of size N by model $\mathbf{\Pi}_t$, i.e. each $\mathbf{X}_t^{(j)} = \left(\mathbf{X}_t^{(j)}(1), \ldots, \mathbf{X}_t^{(j)}(L)\right) \in S = \{0,1\}^L$ is sampled with a probability

$$\prod_{i=1}^{L} \pi_{t,i}^{\mathbf{X}_t^{(j)}(i)} \left[1 - \pi_{t,i}\right]^{1-\mathbf{X}_t^{(j)}(i)}$$

for $j = 1, \ldots, N$.

Memory Update: Build a new memory $\mathbf{M}_{t+1}$ from children $\mathbf{X}_t$ and present memory $\mathbf{M}_t$ according to the fixed memory update rule such that the size $|\mathbf{M}_{t+1}| = M$.

As a further reference, we also write a pseudo code for UMDA in Figure 3.9 on p. 42.

The choice rule here is completely inherited from genetic algorithms. The frequently used rules are: *truncate selection* and *random selection*. In truncate selection, we require the parents size n smaller than the memory size M i.e. $n \leq M$, and we select the n best solutions from memory $\mathbf{M}_t$ as parents population P_t. And in a random selection, each parent in P_t shall be chosen from $\mathbf{M}_t$ by a distribution based on solutions qualities.

The memory update corresponds to the population update in genetic algorithms. There are two commonly used rules: $\lambda + \mu$-rule and (λ, μ)-rule. In a $\lambda + \mu$-rule, we take the M best solutions in $\mathbf{M}_t$ and $\mathbf{X}_t$ as the next memory $\mathbf{M}_{t+1}$. In (λ, μ)-rule, we randomly selection M solutions based on solutions qualities from $\mathbf{M}_t$ and $\mathbf{X}_t$ as the next memory $\mathbf{M}_{t+1}$. Of cause, we may also directly use $\mathbf{X}_t$ as $\mathbf{M}_{t+1}$ in the case $M = N$. And when we use the rule $\mathbf{X}_t = \mathbf{M}_{t+1}$ and truncate selection for parents, then the resulted UMDA becomes a particular CE with $\rho = 1$.

3.4.4 Population based incremental learning

Population-based incremental learning (PBIL), see [Bal94], can be seen as a restricted version of UMDA or a particular version of CE. It selects only the present iteration best

Algorithm Univariate marginal distribution algorithm

a) set t=0, initialize memory $\mathbf{M}_0$ randomly and a stop criterion $STOP$;

b) select a parents population P_t from the present memory by a fixed choice rule;

c) build a model $\mathbf{\Pi}_t = (\pi_{t,1}, \ldots, \pi_{t,L}) \in \mathbb{P}_{um}$ from P_t i.e. each

$$\pi_{t,i} = \frac{\sum_{s=(b_1,\ldots,b_L)\in P_t} b_i}{|P_t|}$$

for $i = 1, \ldots, L$;

d) draw a random sample $\mathbf{X}_t$ by $\mathbf{\Pi}_t$;

e) build the memory $\mathbf{M}_{t+1}$ from $\mathbf{X}_t$ and $\mathbf{M}_t$ by a fixed memory update rule;

f) while $STOP$ does not hold

f1) $t = t + 1$;

f2) repeat steps **b)**-**f)**;

Figure 3.9: Univariate marginal distribution algorithm

solution as parent, i.e. $|P_t| = 1$ and truncate selection. It directly uses the produced children as next memory i.e. $\mathbf{M}_{t+1} = \mathbf{X}_t$. The learning is similar as CE, however an optional feature, called *model mutation*, may apply to the learned model.

Initially, it uses a uniform model $\mathbf{\Pi}_{-1} = (\frac{1}{2}, \ldots, \frac{1}{2}) \in \mathbb{P}_{um}$ to produce the first memory $\mathbf{M}_0$. In each iteration $t \in \mathbb{N}$, the iteration best solution $\mathbf{X}_t^{IT} = \left(\mathbf{X}_t^{IT}(1), \ldots, \mathbf{X}_t^{IT}(L)\right)$ would be picked out from $\mathbf{M}_t$, a model $\mathbf{\Pi}_t = \left(\pi_{t,1}, \ldots, \pi_{t,L}\right) \in \mathbb{P}_{um}$ would be constructed in a way similar as in CE, i.e. for each $i = 1, \ldots, L$,

$$\pi_{t,i} = (1 - \rho)\pi_{t-1,i} + \rho \mathbf{X}_t^{IT}(i) \tag{3.27}$$

where $\rho \in (0, 1)$ is a fixed *learning rate*, and $\mathbf{\Pi}_{t-1} = (\pi_{t-1,1}, \ldots, \pi_{t-1,L}) \in \mathbb{P}_{um}$ is the model for time $t - 1$. Here, observe that an empirical model $\in \mathbb{P}_{um}$ learned from $\{\mathbf{X}_t^{IT}\}$ coincides with $\mathbf{X}_t^{IT}$ itself. After $\mathbf{\Pi}_t$ is constructed, an *optional* model mutation may apply. The model mutation functions similarly as the mutation in genetic algorithms, i.e. both of them may introduce some purely random disturbance into the underlying search so as to help in escaping from local traps. The commonly used model mutation

is the so called *random shift.* With random shift, we mutate $\mathbf{\Pi}_t$ like

$$\pi_{t,i} = \begin{cases} \pi_{t,i}(1 - c_{shift}) & \text{if } u < r_{MR} \text{ and } v < 0.5, \\ \pi_{t,i}(1 - c_{shift}) + c_{shift} & \text{if } u < r_{MR} \text{ and } v > 0.5, \\ \pi_{t,i} & \text{otherwise,} \end{cases} \tag{3.28}$$

for each $i = 1, \ldots, L$, where $c_{shift} \in (0, 1)$ is a fixed shift range, $r_{MR} \in (0, 1)$ is a fixed mutation rate, u and v are independent random variates uniformly distributed over $[0, 1]$. Finally, a number of new solutions are sampled by $\mathbf{\Pi}_t$ as the next memory $\mathbf{M}_{t+1}$. As a reference, a pseudo code of PBIL is listed in Figure 3.10 below.

Algorithm Population-based incremental learning

a) set t=0, initialize $\mathbf{M}_0$ by the uniform distribution $\mathbf{\Pi}_{-1} \in \mathbb{P}_{um}$ and a stop criterion $STOP$;

b) pick out the best solution $\mathbf{X}_t^{IT}$ from $\mathbf{M}_t$;

c) construct $\mathbf{\Pi}_t$ as $\pi_{t,i} = (1-\rho)\pi_{t-1,i} + \rho \mathbf{X}_t^{IT}(i)$ for each $i = 1, \ldots, L$;

d) do model mutation on $\mathbf{\Pi}_t$; %optional

e) draw a new sample $\mathbf{M}_{t+1}$ by $\mathbf{\Pi}_t$;

f) while $STOP$ does not hold

f1) $t = t + 1$;

f2) repeat steps **b)**-**f)**;

Figure 3.10: Population-based incremental learning

3.4.5 Compact genetic algorithm

Compact genetic algorithm (cGA), see [HLG99], is a very simple algorithm which was claimed to efficiently simulate PBIL. In each iteration, it investigates only two feasible solutions and then shifts the distribution very slightly towards the better one. More details, in iteration t, two random solutions $\mathbf{X}_t^{(1)}$ and $\mathbf{X}_t^{(2)}$ are sampled by a present model $\mathbf{\Pi}_t = (\pi_{t,1}, \ldots, \pi_{t,L}) \in \mathbb{P}_{um}$, then it builds a $\mathbf{\Pi}_{t+1} = (\pi_{t+1,1}, \ldots, \pi_{t+1,L})$ for next round as

$$\pi_{t+1,i} = \begin{cases} \pi_{t,i} & \text{if } \mathbf{X}_t^{(1)}(i) = \mathbf{X}_t^{(2)}(i), \\ \pi_{t,i} + \frac{1}{n} & \text{if } \mathbf{X}_t^{(1)}(i) = 1 \neq \mathbf{X}_t^{(2)}(i), \\ \pi_{t,i} - \frac{1}{n} & \text{if } \mathbf{X}_t^{(1)}(i) = 0 \neq \mathbf{X}_t^{(2)}(i) \end{cases} \quad \text{for all } i = 1, \ldots, L, \tag{3.29}$$

where we assume $f(\mathbf{X}_t^{(1)}) \leq f(\mathbf{X}_t^{(2)})$, and n is a large integer fixed in advance. We list cGA in Figure 3.11 below.

Algorithm Compact genetic algorithm

a) set $t = 0$, initialize $\mathbf{\Pi}_0$ and a stop criterion $STOP$;

b) generate two random solutions $\mathbf{X}_t^{(1)}$ and $\mathbf{X}_t^{(2)}$ by $\mathbf{\Pi}_t$;

c) evaluate the two solutions, and to construct $\mathbf{\Pi}_{t+1}$ by

$$\pi_{t+1,i} = \begin{cases} \pi_{t,i} & \text{if } \mathbf{X}_t^{(1)}(i) = \mathbf{X}_t^{(2)}(i), \\ \pi_{t,i} + \frac{1}{n} & \text{if } \mathbf{X}_t^{(1)}(i) = 1 \neq \mathbf{X}_t^{(2)}(i), \\ \pi_{t,i} - \frac{1}{n} & \text{if } \mathbf{X}_t^{(1)}(i) = 0 \neq \mathbf{X}_t^{(2)}(i) \end{cases} \quad \text{for all } i = 1, \ldots, L;$$

d) while $STOP$ does not hold

d 1) $t = t + 1$;

d 2) repeat steps **b)**-**d)**;

Figure 3.11: Compact genetic algorithm

4 A unified model-based search framework

This Chapter serves as a foundation for the next two Chapters. We shall propose a unified MBS framework, and formalize its underlying stochastic process. The framework covers the essential features of the proposed MBS algorithms in Chapter 3. It is actually an extension of the theoretical algorithms investigated in our former work [WK14b] and [WK14a].

To present a unified framework, we first need a (unified) representation for the underlying feasible solutions which may cover the used representations in practical MBS algorithms. Recall that, feasible solutions are represented as fixed length strings over a finite alphabet in CE and EDA, but legal walks on a construction graph in ACO. Here, we shall again employ the former representation, i.e. we represent feasible solutions as *fixed length strings over a finite alphabet* in the unified framework. In the future, we shall refer to this representation as *string encoding* or *string representation*. In Section 4.1, we shall show that string representation actually covers the case of construction graph.

We also need to specify a (unified) model family before presenting the framework. Recall that, the used models families are $\mathbb{P}_{ce}$ (see CE), $\mathbb{P}_{aco}$ (see ACO) and $\mathbb{P}_{um}$ (see EDA). As we choose here the string representation, we shall correspondingly specify $\mathbf{\Pi}_{ce}$ as the model family in the framework. In Section 4.2, we shall show that under the so-called 'feasibility construction', the sampling in the framework can cover sampling methods in CE, EDA and ACO. Here, feasibility construction is a concept extending the visibility in ACO. It can be used to efficiently sample feasible solutions, and introduce mutual dependencies between different positions in a random solution.

With the string representation and model family $\mathbb{P}_{ce}$, we present the unified framework in Section 4.3. The framework obeys the crude MBS procedure on p. 15. To cover essential features of practical MBS algorithms, it has four virtual rules: *memory update rule*, *subsample selection rule*, *learning rule* and *model update rule*. Each rule may have different definitions (implementations) in different MBS algorithms. When we give practical definitions to the four rules, the framework may become a practical MBS algorithm. As a reference, the used rules in practical MBS algorithms are also summarized in Section 4.3.

The framework eventually results in a joint stochastic process. Section 4.4 shall formalize the process, and reveal some basic facts related to the process. We will show the probabilistic dependencies between its marginal processes. In the next two Chapters, we will make a detailed mathematical inspection on the process.

4.1 A unified representation of feasible solutions

Now, we start this Chapter by presenting a unified representation for feasible solutions. This is a necessary pre-step for presenting a unified MBS framework, as we hope the framework can cover essential features of different MBS algorithms. It may facilitate the inspection of the revealed essential features in different algorithms.

As a convention, throughout this Chapter, we shall assume a minimizing CO instance $(S, f, 0)$ where S is the underlying feasible set, f is the objective function and the objective is to find out an optimal solution in S which minimizes f.

4.1.1 Representing feasible solutions as strings

Recall that in CE, we represent feasible solutions as fixed length strings over a finite alphabet. Here, we shall again employ this representation in the framework. Namely, we assume that

$$S \subseteq \mathcal{A}^L = \underbrace{\mathcal{A} \times \mathcal{A} \times \cdot \times \mathcal{A}}_{L} = \{(a_1, \ldots, a_L) \mid a_i \in \mathcal{A}, \quad \text{for each } i = 1, \ldots, L\},$$

where $\mathcal{A}$ is a finite set called *alphabet*, and $L \in \mathbb{N}$ is the fixed *length* of feasible solutions. In the future, we shall refer to this presentation as 'string representation' or 'string encoding'.

Note that in CE, we also assume $S = \mathcal{A}^L$. However, we do not continue this. We allow $S \neq \mathcal{A}^L$. And we call the underlying instance *constrained* if $S \neq \mathcal{A}^L$, otherwise we call it *unconstrained*.

In Subsection 3.2.1, we have seen that MaxCut instances can be unconstrained string encoded. As further examples, we now consider other problems in Section 2.2 bellow.

Example 4.1. *Assume that the underlying CO instance is a TSP instance, see Subsection 2.2.1 in Chapter 2. And let $G = (V, E)$ be the underlying graph. Recall that each $s \in S$ here is a walk on G which starts from a vertex, traverses all other vertices exactly once and ends in the starting vertex, i.e. a Hamiltonian circle. To represent this instance in the above fashion, there are generally two possible methods. The first one is to identify the vertices V as the alphabet $\mathcal{A}$ and the $|V| + 1$ as the string length L, then each feasible solution is represented as a string of length $L = |V| + 1$ over vertices (alphabet) $\mathcal{A} = V$ satisfying that the first position is identical with the last position, and the positions in between can not be identical with any other position. Of cause, under this representation, the underlying instance is constrained since $S \subsetneq \mathcal{A}^L = V^{|V|+1}$. The other one is to represent feasible solutions as binary strings of length n^2 where $n = |V|$. We can label the vertices as $0, 1, 2, \ldots, n-1$. Then, the arcs are labeled as $(0,1), (0,2), \ldots, (0, n-1), \cdots, (n-1, 0), (n-1, 1), \ldots, (n-1, n-1)$. Given a feasible tour (solution), we can represent it as a binary string that we set b_{ij} as 0 if arc (i, j) is on the tour, otherwise we write $b_{ij} = 1$, for all arc (i, j). Under this case, the underlying CO instance is also constrained string encoded with alphabet $\mathcal{A} = \{0, 1\}$ and length $L = n^2$.*

Example 4.2. *Assume that the underlying CO instance is an AP (assignment problem) instance, see Subsection 2.2.2 in Chapter 2. Now, a feasible solution is an assignment of n jobs to m machines where $n \leq m$. Let $\mathcal{J}$ be the collection of jobs, and $\mathcal{M}$ the collection of machines. We label the jobs $\in \mathcal{J}$ as $1, \ldots, n$, and machines $\in \mathcal{M}$ as $1, \ldots, m$. Then, an assignment from $\mathcal{J}$ to $\mathcal{M}$ can be simply represented as a string of length $L = n$ over alphabet $\mathcal{A} = \mathcal{M}$ such that each position i on string represents the label of the corresponding machine to which the i-th job is assigned, for $i = 1, \ldots, n$. Of cause, under this representation, the underlying instance is constrained. And only those strings with unrepeated alphabet are feasible.*

Example 4.3. *Assume that the underlying CO instance is a KP (knapsack problem) instance, see Subsection 2.2.4 in Chapter 2. A feasible solution here is a set of items which are needed to be packed in a knapsack. Let $\mathcal{K}$ be the finite collection of all possible items. A feasible solution is a subset of $\mathcal{K}$ with total weight smaller than a specified capacity of the knapsack. We label items in $\mathcal{K}$ as $1, \ldots, |\mathcal{K}|$, and write $b_i = 1$ if the i-th item is packed in the knapsack and $b_0 = 0$ if the i-th item is not packed. Then a feasible solution can be written as a string of length $L = |\mathcal{K}|$ over alphabet $\mathcal{A} = \{0, 1\}$. Under this representation, the instance is also constrained. The feasible strings are those which have a 'legal' total weight.*

In fact, string representation is very popular in mathematical optimization. For example, in *linear programming* (see [KV02] or Section A. 4 in the Appendix), feasible solutions are generally represented as strings over reals. Actually, string representation may apply to all CO instances. Let $(S', f', 0)$ be an arbitrarily fixed CO instance. We can identify S' as an alphabet. Then, each feasible solution can be seen as a string of length 1 over the alphabet $\mathcal{A} = S'$, although it may not be efficient.

For convenience, we now introduce some standard operations and terminologies of strings which may be frequently used in the future. We use notation $\diamond$ to denote the *empty string* over $\mathcal{A}$. Let $x = (x_1, \ldots, x_m)$ and $y = (y_1, \ldots, y_n)$ be two strings. Then

- $|x| = m$ represents the *length* of string x, particularly $|\diamond| = 0$;
- we write $x \subseteq y$ if string y *extends* string x, i,e, $m \leq n$ and

$$\text{for all } i = 1, \ldots, m, \quad x_i = y_i,$$

  with a convention that $\diamond$ is extended by an arbitrary string;
- we write $a \in x$ for the case that item a is *in* string x i.e.

$$\text{there exists } i = 1, \ldots, m, \quad a = x_i;$$

- we write $(a, b) \in x$ for the case that

$$\text{there exists } i = 1, \ldots, m-1, \quad a = x_i \text{ and } b = x_{i+1};$$

- for any $l \leq |x| = m$, we write $x_{|l}$ for the leading string $(x_1, \ldots, x_l)$ with a convention that $x_{|0} = \diamond$;
- let a be an item , we use (x, a) to mean the *concatenation* of string x with item a, i.e. $(x, a) = \big(x_1, \ldots, x_n, a\big)$.

4.1.2 Relation with other frequently used representations

As a unified representation for MBS algorithms, it is necessary for us to show that the string encoding actually covers other frequently used representations. Recall that, other used representations in MBS algorithms are binary strings (in EDA, see Section 3.4) and construction graph (in ACO, see Section 3.3). Of cause, binary strings are a particular case of the representation here. Thereby, we only need to discuss the case of construction graph.

Recall that in ACO, a feasible solution is represented as a legal walk on a construction graph which starts in a fixed vertex v_0, traverses each of other vertices at most once, and ends in a vertex without unvisited continuations (feasible continuations), see Subsection 3.3.2. Similar with Example 4.1, we may write a legal walk as a string of vertices or arcs. However, when we represent legal walks as strings over vertices, the resulted strings may be of different lengths since the underlying construction graph may not be fully connected. Note that, whether the resulted strings here are of a same length is not essential. We can fill the end of each string with copies of a virtual vertex so as to make them of the same length.

Let $G = (V, E)$ be a construction graph. We now introduce a symbol $\triangle \notin V \cup E$ to the graph as a virtual vertex, and denote the resulted graph as $G' = (V \cup \{\triangle\}, E)$. Note that, a legal walk on G can contain at most $|V|$ vertices. Thereby, the lengths of the resulted strings are at most $|V|$. Let $s = (s_1, \ldots, s_n)$ be the string representation of an arbitrary legal walk on G with $n \leq |V|$ and each $s_i \in V$. Then, s can be equivalently extended as a string of length $|V| + 1$ over vertices on graph G'. Actually, we can add $|V| + 1 - n$ (or $|V| - n$) many copies of the virtual vertex $\triangle$ into the end of s so as to extend it to be a string s' of length $|V| + 1$ over vertices on graph G', i.e.

$$s' = (s_1, \ldots, s_n, \underbrace{\triangle, \triangle, \ldots, \triangle}_{|V|+1-n}),$$

and then assign the same objective value to s'.

In ACO, the feasible solutions are limited to be legal walks on a graph. Of cause, it is also possible that feasible solutions are other kinds of subgraphs. Typically, feasible solutions may be trees, see e.g. the minimum spanning tree problem [KV02]. Here, a tree means a subgraph which is connected but contains no circuit, see also A. 1 in the Appendix. It can be uniquely determined by its arcs. Therefore, we can represent a tree as a binary string by assigning value 1 to those arcs on the underlying graph which belong to the tree, and 0 to the arcs which do not belong to the tree. In more detail, we can first label the arcs E of the underlying graph G as $1, 2, \ldots, |E|$, then a tree can be written as a string $(b_1, \ldots, b_{|E|})$ of length $|E|$ over {0,1} such that $b_i = 1$ means the i-th arc is on the tree, and $b_i = 0$ means the i-th arc is not on tree for $i = 1, \ldots, |E|$.

4.1.3 Introducing constraints under string encoding

In the above, we have shown that string encoding applies to all CO instances, and covers other representations used in MBS algorithms. From Examples 4.1-4.3, we also see that

the underlying CO instance may be constrained under this representation, i.e. $S \neq \mathcal{A}^L$ where S is the underlying feasible set, $\mathcal{A}$ is the alphabet and L is the length of the (encoded) strings. To tell whether a string in $\mathcal{A}^L$ is feasible in the constrained case, we can introduce some *constraints* such that a string is feasible if and only if it satisfies (or models) all the constraints.

As an example, we assume that the underlying instance is a TSP instance, and we employ the vertices V as the alphabet $\mathcal{A}$. By Example 4.1, we can represent each feasible solution as a string of length $|V|+1$ over $\mathcal{A} = V$. Then, a string $s = (s_1, \ldots, s_{|V|+1}) \in V^{|V|+1}$ is feasible if and only if it holds that:

- $s_1 = s_{|V|+1}$;
- for all $i, j = 1, \ldots, |V|, \quad i \neq j \Rightarrow s_i \neq s_j$.

The above two conditions are the introduced constraints.

Now, we use Ω to denote the collection of possible constraints for the underlying CO instance under string encoding, and we set $\Omega = \emptyset$ for the unconstrained case. For a string $s \in \mathcal{A}^L$, we write $s \models \Omega$ to mean that s satisfies all the introduced constraints. Here, we use a convention that $s \models \emptyset$ for every $s \in \mathcal{A}^L$. Then, the underlying feasible set S can written as $\{s \in \mathcal{A}^L \mid s \models \Omega\}$. And the corresponding optimization can be formulated as

$$\begin{array}{l} \min_{s \in \mathcal{A}^L} f\big(s\big) \\ \text{s.t. } s \models \Omega \end{array} \tag{4.1}$$

where 's.t.' shorts for 'subject to'. Note that, (4.1) is actually a general form for CO, since each CO instance can be string encoded.

It is very interesting that with string encoding, we can represent many CO instances as a Linear Programming instance. For example, if we write feasible solutions in a TSP instance as strings over vertices like in Example 4.1 and label the vertices by integers, then resulted optimization in (4.1) shall become a linear (integer) program, namely,

$$\begin{array}{l} \min_{s=(s_1,\ldots,s_{|V|+1}) \in V^{|V|+1}} f(s) = \displaystyle\sum_{i=1}^{|V|} w\big(s_i, s_{i+1}\big) \\ \text{s.t.} \\ \qquad i) \quad s_i - s_j \neq 0 \quad \text{for all } i, j = 1, \ldots, |V| \text{ with } j \neq i, \\ \qquad ii) \quad s_1 - s_{|V|+1} = 0, \end{array}$$

where we label the vertices as $V = \{1, \ldots, |V|\}$, and $w\big(i, j\big)$ is the constant weight on arc (i, j).

4.2 A unified model family and feasibility construction

Recall that, MBS optimize models instead of solutions. Thereby, we still need to specify a (unified) model family before presenting the framework. Here, a necessary criterion for choosing a model family for the framework is that the 'sampling' in various practical MBS algorithms can be formulated as special cases of the sampling by a model from the specified family. Recall that the model families used in MBS algorithms are $\mathbb{P}_{ce}$ in (3.2) on p. 17, the pheromones matrices $\mathbb{P}_{aco}$ in (3.15) on p. 30 and the univariate marginal models $\mathbb{P}_{um}$ in Subsection 3.4.1. With the string representation, we shall again employ $\mathbb{P}_{ce}$ as the model family in the unified framework. However, here we shall not use the sampling in CE, but define another sampling mechanism called *feasibility construction* which is an extension of the visibility in ACO. Note that the feasibility construction is not a new idea, it has been presented in our former work [WK14b] and [WK14a]. In this Section, we shall see that feasibility construction indeed covers those sampling methods in CE, ACO and EDA.

4.2.1 A review to the model family $\mathbb{P}_{ce}$

As formulated in Section 4.1, the underlying feasible set $S \subseteq \mathcal{A}^L$ for some finite alphabet $\mathcal{A}$ and an encoded solutions length $L \in \mathbb{N}$. Recall that, a model $\mathbf{\Pi} \in \mathbb{P}_{ce}$ is a product distribution

$$\big(\mathbf{\Pi}(1), \ldots, \mathbf{\Pi}(L)\big) = \big(\mathbf{\Pi}(a;i)\big)_{a\in\mathcal{A};i=1,\ldots,L}$$

on the product space $\mathcal{A}^L = \mathcal{A} \times \cdots \times \mathcal{A}$ such that each

$$\mathbf{\Pi}(i) = \big(\mathbf{\Pi}(a;i)\big)_{a\in\mathcal{A}}$$

is a distribution on alphabet $\mathcal{A}$. By $\mathbf{\Pi} \in \mathbb{P}_{ce}$, a random string $s = (a_1, \ldots, a_L)$ in $\mathcal{A}^L$ is generated with each position a_i chosen independently by $\mathbf{\Pi}(i)$ for $i = 1, \ldots, L$, i.e. it is chosen with a probability

$$\mathbf{\Pi}(s) = \prod_{i=1}^{L} \mathbf{\Pi}(a_i;i).$$

In the unified framework, we shall employ $\mathbb{P}_{ce}$ as the specified model family. But we do not directly generate random solutions by models in $\mathbb{P}_{ce}$. On the one hand, this may produce infeasible strings (here, $S \neq \mathcal{A}^L$ may hold), although we have discussed in CE that we can remove the constraints by introducing a penalty objective value. On the other hand, the above random solution generating mechanism can not cover the sampling in ACO. To cover the sampling in ACO, we shall use the sampling mechanism proposed in our former works [WK14b] and [WK14a]. We shall call the mechanism as feasibility construction. To show a difference in the sampling with CE, we shall employ the symbol $\mathbb{P}$ to represent the model family $\mathbb{P}_{ce}$ in the future.

4.2.2 A unified sampling mechanism: feasible construction

To propose the feasibility construction, we need some auxiliary definitions. We say that a string $y \in \mathcal{A}^{\leq L} = \{\diamond\} \cup \mathcal{A} \cup \mathcal{A}^2 \cup \cdots \cup \mathcal{A}^L$ is a *feasible partial solution* if there exists a feasible solution $s \in S \subseteq \mathcal{A}^L$ with $y \subseteq s$ (i.e. s extends y). For each $i = 0, \ldots, L$, we use R_i to denote the collection of all feasible partial solutions of length i. Obviously, R_0 is just the singleton $\{\diamond\}$ where recall that $\diamond$ is the empty string. And for each $i = 1, \ldots, L$, the collection R_i can be formally defined as

$$R_i = \big\{(y, a) \big| y \in R_{i-1}, a \in \mathcal{A} \text{ and } \exists s \in S \text{ with } (y, a) \subseteq s\big\} \tag{4.2}$$

where recall that (y, a) indicates the concatenation of partial solution y with item a. Of cause, $S = R_L \subseteq \mathcal{A}^L$, and $R := \bigcup_{i=0}^{L} R_i \subseteq \mathcal{A}^{\leq L}$ is the collection of all possible partial solutions.

For each $i = 0, \ldots, L-1$, we call a function $C_i(\cdot;\cdot) : R_i \times \mathcal{A} \mapsto [0, 1]$ which satisfies that for each $y \in R_i$

- $C_i(y; \cdot)$ is a distribution on $\mathcal{A}$, i.e. $\sum_{a \in \mathcal{A}} C_i(y; a) = 1$,
- for each $a \in \mathcal{A}$, if $(y, a) \notin R_{i+1}$, then $C_i(y; a) = 0$,

as a *feasibility distribution* for position $i+1$. If $C_i(\cdot;\cdot)$ is a feasibility distribution, we let $C_i(y) := \{a \in \mathcal{A} | C_i(y; a) > 0\}$ denote the *support* of $C_i(y; \cdot)$. Obviously, support $C_i(y)$ only contains feasible continuations of y, and $C_i(y; \cdot)$ is a distribution concentrating on those feasible continuations $C_i(y)$.

Note that, for any CO instance we can define a sequence of feasibility distributions $\{C_i(\cdot;\cdot)\}_{i=0}^{L-1}$. For example, we may define a sequence of feasible distributions as

$$C_i(y; a) := \begin{cases} \frac{1}{|\{a \in \mathcal{A} | (y,a) \in R_{i+1}\}|} & \text{for each } a \in \mathcal{A} \text{ with } (y, a) \in R_{i+1}, \\ 0 & \text{otherwise.} \end{cases} \tag{4.3}$$

for $i = 0, \ldots, L-1$ and $y \in R_i$. In the future, we shall refer to (4.3) as the 'non-greedy feasibility distributions'.

We may also introduce some greedy insight into the feasibility distributions. As an example, we consider a TSP instance with weight function w and n vertices in V. And we assume here that feasible solutions (Hamiltonian circles) are represented as strings over vertices. Then, each R_i contains strings with exactly i different vertices. For each $i = 0, \ldots, L-1$ and $y = (v_1, \ldots, v_i) \in R_i$, we can define a feasibility distribution as

$$C_i(y; v) := \begin{cases} \frac{\big[w\big(v_i, v\big)\big]^{-\beta}}{\sum_{(y,v) \in R_{i+1}} \big[w\big(v_i, v\big)\big]^{-\beta}} & \text{if } (y, v) \in R_{i+1}, \\ 0 & \text{otherwise,} \end{cases}$$

where $\beta > 0$ is a fixed constant, $v \in V$ is an arbitrary vertex. Then, the support $C_i(y)$ shall collect the vertices which have not occurred in y yet. And the vertices $\in C_i(y)$ which introduce smaller traveling weights from the end of y (i.e. v_i) would be more preferred as a continuation of y.

Now, we are ready to introduce the feasibility construction method. Feasibility construction is a method inspired by the visibility concept for ACO, see (3.14). It produces a random solution stepwise by a mixture of a production distribution in $\mathbb{P}$ (i.e. $\mathbb{P}_{ce}$) and the feasibility distributions. Let $\mathbf{\Pi} \in \mathbb{P}$ be a product distribution on $\mathcal{A}^L$, and $\{C_i(\cdot;\cdot)\}_{i=0}^{L-1}$ a constructed sequence of feasibility distributions. Starting with the empty string $\diamond$, a partial solution y is then extended by a feasible continuation $a \in C_i(y)$ with a selection probability defined as

$$Q(a; y, i+1, \mathbf{\Pi}) := \frac{\mathbf{\Pi}(a; i+1)C_i(y; a)}{\sum_{a' \in C_i(y)} \mathbf{\Pi}(a'; i+1)C_i(y; a')}, \tag{4.4}$$

where a convention $\frac{0}{0} = 0$ is employed. With (4.4), the selection probability of a complete solution $s = (s_1, \ldots, s_L) \in S$ would be

$$Q\big((s_1, \ldots, s_L); \mathbf{\Pi}\big) := \prod_{i=0}^{L-1} Q\big(s_{i+1}; s_{|i}, i+1, \mathbf{\Pi}\big), \tag{4.5}$$

where recall that $s_{|i} = (s_1, \ldots, s_i)$ and $s_{|0} = \diamond$. Obviously, the support of (4.5) is a subset of S. In other word, by (4.5), we will always produce feasible solutions. As a summary, a pseudo random solution generation algorithm with feasibility construction is listed in Figure 4.1 below.

Algorithm Random solution generation by feasibility construction

a) set $y = \diamond$, and $i = 0$;

b) for $i < L$ do;

b 1) select an item from $\mathcal{A}$ with probability

$$Q(a; y, i+1, \mathbf{\Pi}) := \frac{\mathbf{\Pi}(a; i+1)C_i(y; a)}{\sum_{a' \in C_i(y)} \mathbf{\Pi}(a'; i+1)C_i(y; a')}.$$

b 2) $y = (y, a)$;

b 3) $i = i + 1$;

c) output y;

Figure 4.1: Random solution generation by feasibility construction

Note that, feasibility construction also covers the *unconstrained* case where $S = \mathcal{A}^L$.

In this case, we can set

$$C_i(y;a) = \frac{1}{|\mathcal{A}|} \quad \text{for each } i = 0,\ldots,L-1 \text{ and } y \in R_i. \tag{4.6}$$

Then the $Q(a;y,i+1,\mathbf{\Pi})$ defined in (4.4) is degenerated to $\mathbf{\Pi}(a;i+1)$. Obviously, this will result in a sampling as in CE and EDA. Note also that, in non-greedy feasibility construction (see (4.3)), we take $C_i(y;\cdot)$ as the uniform distribution on set $\{a \in \mathcal{A} | (y,a) \in R_{i+1}\}$, and then

$$Q(a;i+1,y,\mathbf{\Pi}) = Q'(a;i+1,y,\mathbf{\Pi}) := \begin{cases} \frac{\mathbf{\Pi}(a;i+1)}{\sum_{a' \in \mathcal{A},(y,a') \in R_{i+1}} \mathbf{\Pi}(a';i+1)} & \text{if } (y,a) \in R_{i+1} \\ 0 & \text{otherwise.} \end{cases} \tag{4.7}$$

4.2.3 Cover the sampling of ACO

Recall that, in ACO, feasible solutions are represented as legal walks on a construction graph, i.e. those walks starting from a unique vertex, traversing other vertices at most once until no feasible continuations available. Here, we assume a construction graph $G = (V,E)$ with $V = \{v_0, v_1, \ldots, v_K\}$, v_0 is the unique starting vertex for all legal walks. A legal walk on G can be represented as a string over vertices V, or a string over arcs E, and these two representations are equivalent. More details, let

$$v_0, (v_0,a_1), a_1, (a_1,a_2), \ldots, (a_{n-1},a_n), a_n$$

be a legal walk on G, it can be equivalently represented as a string $(v_0, a_1, \ldots, a_n)$ over vertices and a string $\big((v_0,a_1),(a_1,a_2),\ldots,(a_{n-1},a_n)\big)$ over arcs E. Due to equivalence of these three representations, we may use them interchangeably. We will see that the essential samplings in ACO can be formulated as a particular case of the feasibility construction if we write legal walks as strings over arcs.

In ACO, we generate a legal walk by iteratively extending a partial legal with a feasible continuation. Here, a partial legal walk is a walk starting from the unique starting vertex v_0, and traversing other vertices at most once. Let $y = (v_0, a_1, \ldots, a_m)$ be a partial legal walk (here, we write walks as strings over vertices). We use $V_{\text{Feasible}}(y)$ to denote the collection of all feasible continuations to y. Then

$$V_{\text{Feasible}}(y) = \big\{a \in V \mid (a_m,a) \in E,\ a \notin \{v_0,a_1,\ldots,a_m\}\big\}.$$

When $V_{\text{Feasible}}(y) = \emptyset$, y is a (complete) legal walk. Assume $V_{\text{Feasible}}(y) \neq \emptyset$, $\Gamma = \big(\tau(c,d)\big)_{c,d \in V}$ is the present pheromones matrix. Then, an ant may select a feasible continuation $a \in V_{\text{Feasible}}(y)$ to y by the choice probability

$$\mathbf{P}\big[a \mid y\big] = \begin{cases} \frac{\tau(a_m,a)}{\sum_{v' \in V_{\text{Feasible}}(y)} \tau(a_m,v')} & \text{if } a \in V_{\text{Feasible}}(y), \\ 0 & \text{otherwise.} \end{cases} \tag{4.8}$$

Moreover, if there is a visibility information $\eta(y;\cdot)$ available, the choice probability may become

$$\mathbf{P}\big[a \mid y\big] = \begin{cases} \frac{\big[\tau(a_m,a)\big]^{\alpha}\big[\eta(y;a)\big]^{\beta}}{\sum_{v'\in V_{\text{Feasible}}(y)}\big[\tau(a_m,v')\big]^{\alpha}\big[\eta(y;v')\big]^{\beta}} & \text{if } a \in V_{\text{Feasible}}(y), \\ 0 & \text{otherwise,} \end{cases} \tag{4.9}$$

where $\eta(y;v)$ is typically some greedy information like the distance information e.g. $\eta(y;v) = 1/d(a_m,v)$ in TSP, and $\alpha,\beta > 0$ are constants. In application, α is typically chosen to be 1.0, see [DS04], [DMC96], [DG97a], [DG97b], [SH00] etc. Therefore, (4.9) is generally

$$\mathbf{P}\big[a \mid y\big] = \begin{cases} \frac{\tau(a_m,a)\big[\eta(y;a)\big]^{\beta}}{\sum_{v'\in V_{\text{Feasible}}(y)}\tau(a_m,v')\big[\eta(y;v')\big]^{\beta}} & \text{if } a \in V_{\text{Feasible}}(y), \\ 0 & \text{otherwise,} \end{cases} \tag{4.10}$$

in practice.

To show that the feasibility construction covers (4.8) and (4.10), we represent legal walks as equal-length strings over arcs. We use $E \cup \{\triangle\}$ as the underlying alphabet $\mathcal{A}$, where $\triangle$ is a *symbol* not in E. We fill the end of each legal walk with copies of $\triangle$ so as to make them of length $L = |V| - 1$. Note that, each legal walk on the construction graph G can contain at most $|V| - 1$ arcs. According to the pheromones matrix, we can define a product distribution

$$\mathbf{\Pi} = \big(\mathbf{\Pi}(a,b);i\big)_{(a,b)\in\mathcal{A};i=1,\ldots,L} = \big(\mathbf{\Pi}(a,b);i\big)_{(a,b)\in E\cup\{\triangle\};i=1,\ldots,|V|-1}$$

as, for $i = 1,\ldots,L = |V|-1$, and $(a,b) \in \mathcal{A} = E \cup \{\triangle\}$,

$$\mathbf{\Pi}\big((a,b);i\big) = \begin{cases} \frac{p\cdot\tau(a,b)}{\sum_{(c,d)\in E}\tau(c,d)} & \text{if } (a,b) \in E, \\ 1-p & \text{if } (a,b) = \triangle, \end{cases} \tag{4.11}$$

where $p \in (0,1)$ is an arbitrarily fixed constant. Obviously, $\mathbf{\Pi}\big((a,b);1\big) = \cdots = \mathbf{\Pi}\big((a,b);L\big)$ for each item (arc) $(a,b) \in \mathcal{A}$. Each $\mathbf{\Pi}\big(\cdot;i\big)$ indicates a selection in $\mathcal{A} = E \cup \{\triangle\}$ for the i-th position on a random string.

Corresponding to the non-visibility choice probability (4.8), we can construct non-greedy feasibility distributions $\{C_i(\cdot;\cdot)\}_{i=0,\ldots,L-1}$ as following

- for each arc $(a,b) \in \mathcal{A} = E \cup \{\triangle\}$, $C_0\big(\diamond;(a,b)\big) = 0$ if $a \neq v_0$, and

$$C_0\big(\diamond;(a,b)\big) = \frac{1}{|\{b \in V \mid (v_0,b) \in E\}|}$$

  if $a = v_0$,
- for each legal partial walk $Y = \big((v_0,a_1),\ldots,(a_{m-1},a_m)\big)$ (here, we represent walks as strings over arcs), let

$$E_{\text{Feasible}}(Y) = \big\{(a,b) \in E \cup \{\triangle\} \mid a = a_m, b \in V, b \notin \{v_0,a_1,\ldots,a_m\}\big\},$$

and we define, for all $(a,b) \in \mathcal{A} = E \cup \{\triangle\}$,

$$C_m\big(Y;(a,b)\big) = \begin{cases} 1 & \text{if } E_{\text{Feasible}}(Y) = \emptyset, (a,b) = \triangle, \\ 0 & \text{if } E_{\text{Feasible}}(Y) = \emptyset, (a,b) \neq \triangle, \\ \frac{1}{|E_{\text{Feasible}}(Y)|} & \text{if } E_{\text{Feasible}}(Y) \neq \emptyset, (a,b) \in E_{\text{Feasible}}(Y), \\ 0 & \text{otherwise.} \end{cases} \tag{4.12}$$

Then, by the non-greedy feasibility construction (4.7), the probability for choosing an arc $(a,b) \in \mathcal{A}$ as the continuation to a legal partial walk $Y = \big((v_0,a_1),\ldots,(a_{m-1},a_m)\big)$ is

$$Q\big((a,b);Y,m+1,\mathbf{\Pi}\big) = \frac{\mathbf{\Pi}\big((a,b);m+1\big)C_m\big(Y;(a,b)\big)}{\sum_{(c,d)\in\mathcal{A}}\mathbf{\Pi}\big((c,d);m+1\big)C_m\big(Y;(c,d)\big)},$$

where we again use the convention $\frac{0}{0} = 1$. In the case that $V_{\text{Feasible}}(y) \neq \emptyset$, $E_{\text{Feasible}}(Y)$ is also non empty and equals $\{(a_m,b) \mid b \in V_{\text{Feasible}}(y)\}$, where $y = (v_0,a_1,\ldots,a_m)$ is the vertices representation of Y. According to the third and fourth cases in (4.12), the probability

$$\begin{aligned} Q\big((a_m,a);Y,m+1,\mathbf{\Pi}\big) &= \frac{\mathbf{\Pi}\big((a_m,a);m+1\big)C_m\big(Y;(a_m,a)\big)}{\sum_{c\in V_{\text{Feasible}}(y)}\mathbf{\Pi}\big((a_m,c);m+1\big)C_m\big(Y;(a_m,c)\big)} \\ &= \frac{\mathbf{\Pi}\big((a_m,a);m+1\big)\frac{1}{|E_{\text{Feasible}}(Y)|}}{\sum_{c\in V_{\text{Feasible}}(y)}\mathbf{\Pi}\big((a_m,c);m+1\big)\frac{1}{|E_{\text{Feasible}}(Y)|}} \\ &= \frac{\mathbf{\Pi}\big((a_m,a);m+1\big)}{\sum_{c\in V_{\text{Feasible}}(y)}\mathbf{\Pi}\big((a_m,c);m+1\big)}, \end{aligned}$$

for a vertex $a \in V_{\text{Feasibility}}(y)$. By the first case in (4.11), we have

$$\begin{aligned} Q\big((a_m,a);Y,m+1,\mathbf{\Pi}\big) &= \frac{\frac{p\cdot\tau(a_m,a)}{\sum_{(h,d)\in E}\tau(h,d)}}{\sum_{c\in V_{\text{Feasible}}(y)}\frac{p\cdot\tau(a_m,c)}{\sum_{(h,d)\in E}\tau(h,d)}} \\ &= \frac{\tau(a_m,a)}{\sum_{c\in V_{\text{Feasible}}(y)}\tau(a_m,c)}, \end{aligned}$$

which is exactly the same as the first case in the choice probability (4.8) for non-visibility. When $a \notin V_{\text{Feasibility}}(y)$, we can similarly show that the probability is 0. And if $V_{\text{Feasibility}}(y) = \emptyset$ i.e. y is a complete legal walk, then $E_{\text{Feasibility}}(Y)$ is also empty. In this case, the probability for choosing $\triangle$ is 1 provided that Y contains less than L arcs. So, we deterministically add copies of $\triangle$ to the end of a 'complete' Y until it contains L arcs.

For covering the visibility case (4.10), we only need to redefine $C_0\big(\diamond;(v_0,a)\big)$ as

$$\frac{\big[\eta((v_0);a)\big]^\beta}{\sum_{c\in V_{\text{Feasibility}}(y)}\big[\eta((v_0);c)\big]^\beta}$$

where (v_0) means the string only consisting of v_0, and the feasibility distributions (4.12) as

$$C_m\big(Y;(a,b)\big) = \begin{cases} 1 & \text{if } E_{\text{Feasible}}(Y) = \emptyset, (a,b) = \triangle, \\ 0 & \text{if } E_{\text{Feasible}}(Y) = \emptyset, (a,b) \neq \triangle, \\ \dfrac{\big[\eta(y;b)\big]^{\beta}}{\sum_{c \in V_{\text{Feasibility}}(y)} \big[\eta(y;c)\big]^{\beta}} & \text{if } E_{\text{Feasible}}(Y) \neq \emptyset, (a,b) \in E_{\text{Feasible}}(Y), \\ 0 & \text{otherwise,} \end{cases}$$

where $Y = \big((v_0,a_1),\ldots,(a_{m-1},a_m)\big)$, $y = (v_0,v_1,\ldots,v_m)$ is its vertices representation. The verification is similar as in the case of non-visibility.

4.2.4 Probabilistic dependencies in the sampling

In the above discussion, we have seen that feasibility construction covers the samplings used in CE, ACO and EDA. Therefore, by feasibility construction, it is 'legal' to use $\mathbb{P} = \mathbb{P}_{ce}$ as the model family in the unified framework. Actually, feasibility construction may also introduce some 'crude' probabilistic dependencies into the sampling.

Recall that, if we generate a random solution $s = (a_1,\ldots,a_L)$ only by a model $\mathbf{\Pi} = \big(\mathbf{\Pi}(a;i)\big)_{a\in\mathcal{A};i=1,\ldots,L} \in \mathbb{P}$, then positions on s are generated independently i.e.

$$\mathbf{P}\big[a_i \;\big|\; (a_1,\ldots,a_{i-1})\big] = \mathbf{P}\big[a_i\big] = \mathbf{\Pi}(a_i;i)$$

for each $i = 1,\ldots,L$, where we use a convention that $(a_1,a_0) = \diamond$. In feasibility construction, the next item may depend on the finished leading string, i.e.

$$\mathbf{P}\big[a_i \;\big|\; (a_1,\ldots,a_{i-1})\big] = Q\big(a_i;(a_1,\ldots,a_{i-1}),i,\mathbf{\Pi}\big) \neq \mathbf{P}[a_i]$$

in the constrained case.

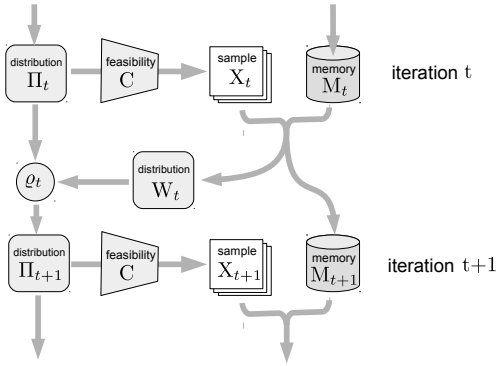
Actually, feasibility construction can be made more extensive so as to cover some linkage learning mechanism [PM98] into the sampling. However, we will not discuss this issue here, we leave it as a future work.

4.3 A unified model-based search framework

In Sections 4.1 and 4.2, we have fixed the string representation as the unified representation and $\mathbb{P} = \mathbb{P}_{ce}$ as the unified model family. Now, we are ready to present the unified framework. Here, we emphasize once more that the underlying CO instance is $(S, f, 0)$ with $S = \mathcal{A}^L$ for some alphabet $\mathcal{A}$ and a fixed solutions length $L \in \mathbb{N}$, the model family is $\mathbb{P} = \mathbb{P}_{ce}$, and the feasibility construction is employed in the sampling i.e we sample random solutions with algorithm in Figure 4.1.

4.3.1 The unified framework for model-based search

The picture below visually describes the unified MBS framework we are going to present.



In the above picture, $\mathbf{\Pi}_t$ and C represent the model in iteration t and the constructed feasibility distributions respectively. The $\mathbf{X}_t$ is a random sample produced by feasibility construction (4.5).

The $\mathbf{M}_t$ in the picture is called *memory* which records some solutions sampled before the present iteration t with a convention that $\mathbf{M}_0 = \emptyset$. Typically, $\mathbf{M}_t$ may contain some best solutions, e.g. the best so far solution as in ACS and $\mathcal{MMAS}$. After $\mathbf{X}_t$ is sampled, the next memory $\mathbf{M}_{t+1}$ is constructed with a specified *memory update rule*. Note that some MBS algorithms may not use memories, e.g. CE. In this case, we always identify the memory as $\emptyset$.

The $\mathbf{W}_t$ is the *empirical distribution* learned from $\mathbf{X}_t \cup \mathbf{M}_t$.[1] Generally, we may learn $\mathbf{W}_t$ from a subsample $\mathcal{N}_t^b$ of $\mathbf{X}_t \cup \mathbf{M}_t$ with a specified *learning rule*. And a rule for selecting $\mathcal{N}_t^b$ is called a *selection rule*. Note that, the selection is usually biased to good solutions. For instance, in CE, we learn the empirical distribution from a subsample $\mathcal{N}_t^b$ which consists of elite solutions in $\mathbf{X}_t$, i.e. a truncate selection rule is employed.

The next model $\mathbf{\Pi}_{t+1}$ is constructed from current model $\mathbf{\Pi}_t$ and the learned empirical distribution $\mathbf{W}_t$ with a specified *distribution update rule*. In MBS, we often construct $\mathbf{\Pi}_{t+1}$ as

$$\mathbf{\Pi}_{t+1} = (1 - \rho_{t+1})\mathbf{\Pi}_t + \rho_{t+1}\mathbf{W}_t, \tag{4.13}$$

[1] In this Thesis, we identify samples and memories as vectors or strings of solutions. And the notation $\mathbf{X}_t \cup \mathbf{M}_t$ is equivalent to $(\mathbf{X}_t, \mathbf{M}_t)$ which means the concatenation of $\mathbf{X}_t$ and $\mathbf{M}_t$.

where ρ_{t+1} is a *learning rate*. Obviously, learning rates correspond to the smooth parameters in CE and the evaporation rates in ACO. In the sequel, we shall concentrate on this kind of update, and we shall refer to (4.13) as the 'basic recursion'. In update (4.13), the learning rate ρ_{t+1} reflects the relative importance of the empirical distribution $\mathbf{W}_t$.

Now, we list the unified framework for MBS algorithms in Figure 4.2 below. Obviously, this framework covers the essential features of CE, ACO and EDAs. Note that, this framework itself is an optimization algorithm with four components, namely memory update rule, selection rule, learning rule and possibly the distribution update rule. Later, we shall summarize the used rules in the practical MBS algorithms. The mathematical study in Chapters 5 and 6 shall be based on this framework.

Algorithm A unified model-based search framework

Elements:

an instance (S, f) with $S \subseteq \mathcal{A}^L$ for some finite set $\mathcal{A}$ and $L \in \mathbb{N}$, a sample size $N \in \mathbb{N}$, a sequence of learning rates $(\rho_t)_{t\geq 1}$ on $(0, 1]$, an initial distribution $\mathbf{\Pi}_0 \in \mathbb{P}$, a sequence of feasibility distributions $C_i(\cdot;\cdot)$ for $i = 0, \ldots, L-1$, a memory update rule, a subsample selection rule, a learning rule, and a stop criterion $STOP$.

algorithm:

a) set $\mathbf{M}_0 = \emptyset$ and $t = 0$;

b) do while $STOP$ does not hold;

b 1) draw a sample $\mathbf{X}_t$ using algorithm in Figure 4.1 on p. 52 with $\mathbf{\Pi} = \mathbf{\Pi}_t$ and $C_i(\cdot;\cdot)$'s;

b 2) construct $\mathbf{M}_{t+1}$ from $\mathbf{M}_t$ and $\mathbf{X}_t$ with the memory update rule;

b 3) select a subsample $\mathcal{N}_t^b$ from $\mathbf{M}_t \cup \mathbf{X}_t$ with the subsample selection rule;

b 4) learn a distribution $\mathbf{W}_t \in \mathbb{P}$ from $\mathcal{N}_t^b$ with the learning rule;

b 5) $\mathbf{\Pi}_{t+1} = (1 - \rho_{t+1})\mathbf{\Pi}_t + \rho_{t+1}\mathbf{W}_t$;

Figure 4.2: A unified model-based search framework for combinatorial optimization

4.3.2 A summary of commonly-used rules in practical model-based search algorithms

From Sections 4.1 and 4.2, we see that the sampling method in the framework covers the sampling methods in practical MBS algorithms. For covering the essential features of those practical algorithms, we now summarize the commonly used rules for the four components from the practical MBS algorithms in Chapter 3.

Memory update rules

Memory update rule tells which solutions should be stored for a possible use in next round. By a memory update rule, we can construct the next memory $\mathbf{M}_{t+1}$ from the present memory $\mathbf{M}_t$ and sample $\mathbf{X}_t$. Usually, this rule may affect the empirical distribution in next round, because $\mathbf{W}_t$ is learning from $\mathbf{M}_t \cup \mathbf{X}_t$. In general, there are three kinds of memory update rules. Here, we call them non memory (NM), globally truncate memory update (GTMU) and locally truncate memory update (LTMU) respectively.

In NM, $\mathbf{M}_t \equiv \emptyset$ for all $t \in \mathbb{N}$, i.e we do not store the feasible solutions seen previously. Typical examples of MBS algorithms with NM, are the basic CE (cross entropy algorithm, see [Rub99]), CE/tdsp (cross entropy algorithm with time-dependent smooth parameters, see [CJK07]), CE/as (ant-like cross entropy algorithm, see [WK14b]), AS (ant system, see [DMC96]), cGA (compact genetic algorithm, see [HLG99]), UMDA (univariate marginal distribution algorithm, see [PM98]), and PBIL (population-based incremental learning, see [Bal94]).

In GTMU, the next memory $\mathbf{M}_{t+1}$ consists of some best solutions in $\mathbf{M}_t$ and $\mathbf{X}_t$. In details, we order solutions in $\mathbf{M}_t \cup \mathbf{X}_t$ as

$$f(s^{(n_1)}) \leq f(s^{(n_2)}) \leq \ldots \leq f(s^{(n_{M+N})})$$

where N is the size of $\mathbf{X}_t$ and M is the size of $\mathbf{M}_t$, and then set

$$\mathbf{M}_{t+1} = (s^{(n_1)}, \ldots, s^{(n_M)}). \tag{4.14}$$

MBS algorithms with GTMU are e.g. ACS (ant colony system, see [DG97b]), GBAS (graph-based ant system, see [Gut00]), and GBCE (graph-based cross entropy algorithm, see [Mar05]) and UMDA.

In LTMU, a fixed *out* rule is employed to remove a number of solutions from $\mathbf{M}_t$, then $\mathbf{M}_{t+1}$ shall consist of the same number of best solutions in $\mathbf{X}_t$ as well as the remaining solutions in $\mathbf{M}_t$. In details, let $O = (x^{(1)}, \ldots, x^{(k)}) \subseteq \mathbf{M}_t$ be k solutions determined by an out rule, and $I = (y^{(1)}, \ldots, y^{(k)})$ the k best solutions in $\mathbf{X}_t$. Then

$$\mathbf{M}_{t+1} = \big[\mathbf{M}_t - O\big] \cup I.^2 \tag{4.15}$$

Note that, we may keep the out rule disabled in the first several iterations until the memory has reached a specified size. Typical examples of out rules are worst out (WO) and first in first out (FIFO). In WO, we kick out a fix amount of worst solutions. In FIFO, we kick out a number of oldest solutions. A type example of LTMU is PBAS (population-based ant system [GM02]).

[2]Here $\mathbf{M}_t - O$ is the string which is formed by removing O from $\mathbf{M}_t$.

4.3.3 Selection rules

In MBS, the empirical distribution $\mathbf{W}_t$ is typically learned from a subsample $\mathcal{N}_t^b$ of $\mathbf{X}_t \cup \mathbf{M}_t$. A selection rule may tell how to select $\mathcal{N}_t^b$ in each iteration t. This rule may directly affect the empirical model $\mathbf{W}_t$. In practice, selection rules can be collected as identity selection (ID), memory identical selection (MID), truncate selection (TS), random selection (RS) and memory random selection (MRS).

In ID selection, we set $\mathcal{N}_t^b = \mathbf{M}_t \cup \mathbf{X}_t$. MBS algorithms with ID selection are e.g. AS (ant system, see [DMC96]) and cGA (compact genetic algorithm, see [HLG99]).

In MID selection, we set $\mathcal{N}_t^b = \mathbf{M}_{t+1}$. Note that, $\mathbf{M}_{t+1}$ is also a subsample of $\mathbf{M}_t \cup \mathbf{X}_t$. Typical algorithms with MID selection are e.g. ACS (ant colony system, [DG97b]), PBAS (population-based ant system, [GM02]), GBAS (graph-based ant system, [Gut00]) and GBCE (graph-based cross entropy, [Mar05]).

TS selection is completely similar with the GTMU memory update. It collects a fixed number of best solutions in $\mathbf{M}_t \cup \mathbf{X}_t$ as $\mathcal{N}_t^b$. MBS algorithms with TS selection are e.g. CE (cross entropy algorithm, [RK04]), CE/tdsp (cross entropy with time-dependent smooth parameters, see [CJK07]), CE/as (ant-like cross entropy algorithm, [WK14b]), UMDA (univariate marginal distribution algorithm, [MP96]) and PBIL (population-base incremental learning, [Bal94]).

In the case of RS, we select $\mathcal{N}_t^b$ from $\mathbf{X}_t \cup \mathbf{M}_t$ by probabilities proportional to qualities of solutions, i.e.

$$\mathbf{P}\left[s \in \mathcal{N}_t^b \;\middle|\; s \in \mathbf{M}_t \cup \mathbf{X}_t\right] = \frac{g(f(s))}{\sum_{s' \in \mathbf{M}_t \cup \mathbf{X}_t} g(f(s'))} \quad \text{for all } s \in \mathbf{M}_t \cup \mathbf{X}_t, \tag{4.16}$$

where $g(\cdot)$ is a fixed positive non-increasing function. Examples of RS selection can be found in UMDA and some other EDAs (see Hauschild et al[HP11]).

MRS selection is similar with RS selection. However, here $\mathcal{N}_t^b$ is sampled from $\mathbf{M}_{t+1}$ i.e.

$$\mathbf{P}\left[s \in \mathcal{N}_t^b \;\middle|\; s \in \mathbf{M}_t \cup \mathbf{X}_t\right] = \frac{g(f(s))}{\sum_{s' \in \mathbf{M}_{t+1}} g(f(s'))} \quad \text{for all } s \in \mathbf{M}_{t+1}, \tag{4.17}$$

where $g(\cdot)$ is a fixed positive non-increasing function. Algorithm with MRS is e,g, $\mathcal{MMAS}$ [SH00]

4.3.4 The learning rules

Given $\mathcal{N}_t^b$, an empirical model $\mathbf{W}_t \in \mathbb{P} = \mathbb{P}_{ce}$ shall be learned by a learning rule. In MBS algorithms, popular learning rules are uniform learning (UL) and weighted learning (WL).

Typical examples of UL may be CE [Rub99], UMDA [MP96] and PBIL [Bal94]. With UL, we see the solutions in $\mathcal{N}_t^b$ equivalently, and use the relative frequencies as the empirical distribution $\mathbf{W}_t$, i.e.

$$\mathbf{W}_t(a;i) := \frac{\sum_{s \in \mathcal{N}_t^b} \mathbb{1}_{\{a\}}(s_i)}{|\mathcal{N}_t^b|} \text{ for all } a \in \mathcal{A} \text{ and } i = 1, \ldots, L. \tag{4.18}$$

By WL, we see the solutions inequivalently, and use the weighted frequencies as $\mathbf{W}_t$. Let $g(\cdot)$ be a positive real function satisfying $g(s^{(1)}) \geq g(s^{(2)}) \iff f(s^{(1)}) \leq f(s^{(2)})$. Then, we may calculate the empirical distribution as

$$\mathbf{W}_t(a;i) := \frac{\sum_{s\in\mathcal{N}_t^b} \mathbb{1}_{\{a\}}(s_i)g(s)}{\sum_{s\in\mathcal{N}_t^b} g(s)} \text{ for all } a \in \mathcal{A} \text{ and } i = 1,\ldots,L. \tag{4.19}$$

Algorithms with WL are various kinds of ACO algorithms (see [DS04]).

Of cause, we may make the learning phase more flexible, i.e.

$$\mathbf{W}_t(a;i) := \frac{\sum_{s\in\mathcal{N}_t^b} \mathbb{1}_{\{a\}}(s_i)g(s,t)}{\sum_{s\in\mathcal{N}_t^b} g(s,t)} \text{ for all } a \in \mathcal{A} \text{ and } i = 1,\ldots,L, \tag{4.20}$$

for some fixed *positive* function g with $g(s^{(1)},t) \geq g(s^{(2)},t) \iff f(s^{(1)}) \leq f(s^{(2)})$ for $t \in \mathbb{N}$. We shall refer to this kind of learning as time dependent weighted learning (TDWL) in the sequel.

4.3.5 Distribution update rules

With the present model $\mathbf{\Pi}_t$ and the empirical model $\mathbf{W}_t$, we can construct the model $\mathbf{\Pi}_{t+1}$ for next round according to a fixed distribution update rule. MBS algorithms generally employ the basic recursion (4.13) to form next model. According to the range of $(\rho_t)_{t\geq 1}$, we can collect MBS into two classes, namely, *conservative* and *radical*.

Conservative distribution update is popular in MBS. It restricts each learning rate in $(0,1)$. For examples, see CE [RK04], ACO [DS04] and EDAs using univariate marginal distributions [HP11] except for UMDA (univariate marginal distribution algorithm, [PM98]).

Radical distribution update occurs only in the UMDA [PM98]. It directly uses the empirical distribution $\mathbf{W}_t$ as the next model $\mathbf{M}_{t+1}$, in other word $\rho_t \equiv 1$.

Actually, we can make the update of distributions in the framework more extensively by formulating it as a (possibly random) function $\mathcal{U} : \mathbb{P} \times \mathbb{P} \mapsto \mathbb{P}$. And if a definition of $\mathcal{U}$ is given, we can set the next model $\mathbf{\Pi}_{t+1}$ as $\mathcal{U}(\mathbf{\Pi}_t, \mathbf{W}_t)$. In MBS, the basic recursion (4.13) is the most popular definition for $\mathcal{U}$. Of cause, we can employ other definitions. For example, in PBIL (population-based incremental learning), we can define the $\mathcal{U}$ as a composition of the basic recursion and the additional model mutation function.

Although we can make the update of models more extensive, we will still stick to (4.13) in the future, because of its popularity.

As a summary to this Section, Table 4.1 on p. 62 shows the used rules in practical MBS algorithms.

Table 4.1: The four rules for some practical MBS algorithms

algorithms	memory update	selection rule	learning rule	distr. update
CE [Rub99]	NM	TS	UL	$\rho_t \equiv \rho \in (0,1)$
CE/as [WK14b]	NM	TS	UL	$\rho_t \in (0,1)$
GBCE [Mar05]	GTMU	MID	UL	$\rho_t \in (0,1)$
CE/tdsp [CJK07]	NM	TS	UL	$\rho_t \in (0,1)$
AS	NM	ID	WL	$\rho_t \equiv \rho = 1$
ACS	GTMU	MID	WL	$\rho_t \equiv \rho \in (0,1)$
$\mathcal{MMAS}$	LTMU (WO)	MRS	WL	$\rho_t \equiv \rho \in (0,1)$
PBAS [GM02]	LTMU (FIFO)	MID	WL	$\rho_t \equiv \rho = 1$
GBAS [Gut00]	GTMU	MID	WL	$\rho_t \equiv \rho \in (0,1)$
1-ANT [NW06]	GTMU	ID	WL	$\rho_t \equiv \rho \in (0,1)$
AS_{elite}[BHS97]	GTMU	ID	WL	$\rho_t \equiv \rho \in (0,1)$
AS_{rank}[BHS97]	GTMU	ID	WL	$\rho_t \equiv \rho \in (0,1)$
AS_{bw}[CdVHM00]	GTMU	ID	WL	$\rho_t \equiv \rho \in (0,1)$
UMDA	NM or GTMU	TS or RS	UL	$\rho_t \equiv \rho = 1$
PBIL	NM	TS	UL	$\rho_t \equiv \rho \in (0,1)$
cGA	NM	ID	UL	$\rho_t \equiv \rho \in (0,1)$
EDAs [ZM04]	NM	TS or RS	UL	$\rho_t \equiv \rho = 1$

4.4 The underlying stochastic process

In Section 4.3, we have presented the unified framework for MBS algorithms based on the string representation. The framework specifies $\mathbb{P} = \mathbb{P}_{ce}$ as model family, and employs the feasibility construction in sampling. It iteratively evolves models in $\mathbb{P}$ according to four rules i.e. a memory update rule, a selection rule, a learning rule and a distribution update rule. Due to this evolution, it eventually results in a joint stochastic process. As an end to this Chapter, this Section shall formally define the process.

4.4.1 Input parameters and strategy

The framework generally takes the following as input parameters (see also Figure 4.2):

- a fixed *initial model* $\mathbf{\Pi}_0 \in \mathbb{P}$;
- a fixed *feasibility distributions* sequence $\{C_i(\cdot;\cdot)\}_{i=0}^{L-1}$;
- a fixed *learning rates* sequence $(\rho_t)_{t=1,2,\ldots}$ with each $\rho_t \in (0, 1]$;
- a fixed *sample size* $N \in \mathbb{N}$;
- a fixed *memory size* $M \in \mathbb{N}$;
- a fixed *subsample size* $N_b \in \mathbb{N}$;
- a fixed *memory update rule* $\mathcal{M}$;
- a fixed *subsample selection rule* $\mathcal{S}$;
- a fixed *learning rule* $\mathcal{L}$.

Note that, we restrict the memory update rule to be the basic recursion

$$\mathbf{\Pi}_{t+1} = (1 - \rho_{t+1})\mathbf{\Pi}_t + \rho_{t+1}\mathbf{W}_t.$$

Note also that, the learning rate ρ_t is required to be *positive*. The main reason is that practical MBS algorithms generally employ positive learning rates, see Table 4.1.

A *strategy* of the framework is then a combination of a fixed feasibility distributions sequence, a fixed learning rates sequence, a fixed memory update rule, a fixed subsample selection rule and a fixed learning rule. With a specified strategy, the framework may correspond to a particular MBS algorithm. For example, if we use the unconstrained feasibility distributions (4.6), fix a *constant learning rate* $\rho_t \equiv \rho > 0$, and employ resp., rules NM (non memory), TS (truncate selection) and UL (uniform learning) for memory update, subsample selection and learning, the framework then becomes a CE algorithm (see Table 4.1). In the sequel, we use notation "$\mathcal{M} =$" to short for the words "memory update rule is", similarly for the notation "$\mathcal{S} =$" and "$\mathcal{L} =$". As an example, "$\mathcal{M} =$ NM", "$\mathcal{S} =$ TS" and "$\mathcal{L} =$ UL" correspond resp., to the sentences "memory update rule is non-memory", "subsample selection rule is truncate selection" and "learning rule is uniform learning".

4.4.2 The underlying stochastic process

With a fixed strategy, the framework will iteratively evolve models $\in \mathbb{P} = \mathbb{P}_{ce}$. This would result in a joint stochastic process (see A. 4 in the Appendix for a definition of

stochastic process)

$$\left(\mathbf{\Pi}_t; \mathbf{X}_t; \mathbf{M}_t; \mathcal{N}_t^b; \mathbf{W}_t\right)_{t=0,1,2,\ldots}.$$

Where $\mathbf{\Pi}_t \in \mathbb{P}$ represents the model in iteration t which is a product distribution on the product space $\mathcal{A}^L = \mathcal{A} \times \cdots \times \mathcal{A}$; $\mathbf{X}_t \in S^N$ is a random sample drawn by a mixture of $\mathbf{\Pi}_t$ and the feasibility distributions $\{C_i\}_{i=0}^{L-1}$ with probabilities defined in (4.5); $\mathbf{M}_t \in S^M$ is the memory maintained by a fixed rule $\mathcal{M}$, which records some 'best' solutions seen in times $0, 1, 2, \ldots, t-1$; $\mathcal{N}_t^b \in S^{N_b}$ is a subsample of $\mathbf{X}_t \cup \mathbf{M}_t$ selected by a fixed rule $\mathcal{S}$; $\mathbf{W}_t \in \mathbb{P}$ is an 'empirical' distribution learned from $\mathcal{N}_t^b$ by a fixed rule $\mathcal{L}$. Initially, a starting model $\in \mathbb{P}$ is determined i.e. $\mathbf{\Pi}_0$ is fixed, and the memory is set to be an empty vector i.e. $\mathcal{M}_0 = \emptyset$. Then, the process iteratively evolves models as in Figure 4.3 below. The thin dashed (gray) line indicates the subsample selection in the case that $\mathcal{S} = \text{MID}$

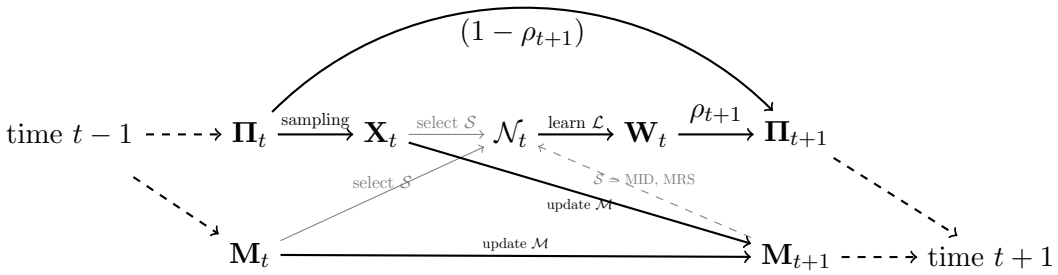


Figure 4.3: Underlying process

or MRS (memory identity selection or memory random selection), and the thin (gray) line for other cases.

In the above Figure, $\mathbf{\Pi}_t = (\mathbf{\Pi}_t(1), \ldots, \mathbf{\Pi}_t(L))$ is the model for iteration t, where each $\mathbf{\Pi}_t(i) = \Big(\mathbf{\Pi}_t(a;i)\Big)_{a\in\mathcal{A}}$ describes a distribution on the fixed alphabet $\mathcal{A}$ i.e.

$$\sum_{a\in\mathcal{A}} \mathbf{\Pi}(a;i) = 1 \quad \text{and} \quad \mathbf{\Pi}(a;i) \geq 0 \text{ for each } a \in \mathcal{A}$$

for $i = 1, \ldots, L$. The state space $\mathbb{P} = \mathbb{P}_{ce}$ of the marginal process $\left(\mathbf{\Pi}_t\right)_{t=0,1,2,}$ is continuous, since the state space $\mathbb{P}(\mathcal{A})$ of the process $\left(\mathbf{\Pi}_t(i)\right)_{t=0,1,2,\ldots}$ is continuous for each $i = 1, \ldots, L$.

By the feasibility construction (4.5), we can draw a random sample

$$\mathbf{X}_t = \left(\mathbf{X}_t^{(1)}, \ldots, \mathbf{X}_t^{(N)}\right)$$

where each

$$\mathbf{X}_t^{(l)} = \left(\mathbf{X}_t^{(l)}(1), \ldots, \mathbf{X}_t^{(l)}(L)\right) \in S \subseteq \mathcal{A}^L$$

for $l = 1, \ldots, N$, by $\mathbf{\Pi}_t$ and feasibility distributions. In details, for each $l = 1, \ldots, N$ and each $i = 0, \ldots, L-1$, $\mathbf{X}_t^{(l)}(i+1)$ is sampled by a conditional probability

$$Q\big(\mathbf{X}_t^{(l)}(i{+}1); \mathbf{X}_t^{(l)}(1, \ldots, i), i{+}1, \mathbf{\Pi}_t\big) = \frac{\mathbf{\Pi}_t\big(\mathbf{X}_t^{(l)}(i+1); i+1\big) C_i\big(\mathbf{X}_t(1, \ldots, i); \mathbf{X}_t^{(l)}(i+1)\big)}{\sum_{a \in C_i\left(\mathbf{X}_t^{(l)}(1,\ldots,i)\right)} \mathbf{\Pi}_t\big(a; i+1\big) C_i\big(\mathbf{X}_t(1, \ldots, i); a\big)}$$

where $\mathbf{X}_t^{(l)}(1, \ldots, i) = \mathbf{X}_t^{(l)}{}_{|i} = \big(\mathbf{X}_t^{(l)}(1), \ldots, \mathbf{X}_t^{(l)}(i)\big)$ is the *leading partial solution* up to length i in $\mathbf{X}_t^{(l)}$, and we use the convention that $\mathbf{X}_t^{(l)}{}_{|0} = \diamond$. Therefore, for each $\mathbf{x} \in S^N$,

$$\mathbf{P}\big[\mathbf{X}_t = \mathbf{x} \;\big|\; \mathbf{\Pi}_t, \mathcal{H}_{t-1}\big] = \mathbf{P}\big[\mathbf{X}_t = \mathbf{x} \;\big|\; \mathbf{\Pi}_t\big],$$

for each $t \geq 0$, where $\mathcal{H}_{t-1}$ is the history of the joint process up to time $t-1$ i.e.

$$\mathcal{H}_{t-1} = \big\{\mathbf{\Pi}_0, \mathbf{X}_0, \mathbf{M}_0, \mathcal{N}_0^b, \mathbf{W}_0; \cdots; \mathbf{\Pi}_{t-1}, \mathbf{X}_{t-1}, \mathbf{M}_{t-1}, \mathcal{N}_{t-1}^b, \mathbf{W}_{t-1}\big\},$$

and we use a convention that $\mathcal{H}_{-1} = \emptyset$. Obviously, the state space $\mathcal{A}^L$ of marginal process $\big(\mathbf{X}_t\big)_{t=0,1,2\ldots}$ is S^N which is finite.

The state space of the marginal process $\big(\mathbf{M}_t\big)_{t=0,1,2,\ldots}$ is $S^M \cup \{\emptyset\}$ which is also finite. Initially, $\mathbf{M}_0 = \emptyset$. Then, the memory $\mathbf{M}_t \in S^M$ is a *vector* containing M solutions for each iteration $t \geq 1$. In each iteration $t \in \mathbb{N}$, the memory shall be updated by the fixed memory rule $\mathcal{M}$. By the rule, the next memory $\mathbf{M}_{t+1}$ is then a subsample of $\mathbf{X}_t \cup \mathbf{M}_t$. For example, if $\mathbf{M} =$ GTMU (global truncate memory update rule), we shall take the best M solutions in present sample $\mathbf{X}_t$ and present memory $\mathbf{M}_t$ as the next memory $\mathbf{M}_{t+1}$. Obviously, for each $\mathbf{m} \in S^M$,

$$\mathbf{P}\big[\mathbf{M}_{t+1} = \mathbf{m} \;\big|\; \mathbf{X}_t, \mathbf{\Pi}_t, \mathbf{M}_t, \mathcal{N}_t^b, \mathbf{W}_t, \mathcal{H}_{t-1}\big] = \mathbf{P}\big[\mathbf{M}_{t+1} = \mathbf{m} \;\big|\; \mathbf{X}_t, \mathbf{M}_t\big],$$

for each $t \in \mathbb{N}$.

The marginal process $\big(\mathcal{N}_t^b\big)_{t=0,1,2,\ldots}$ also has a finite state space i.e. S^{N_b}. Each $\mathcal{N}_t^b$ is a subsample of $\mathbf{X}_t \cup \mathbf{M}_t$ chosen by the fixed selection rule $\mathcal{S}$. When $\mathcal{S} =$ MID (memory identity selection) or $\mathcal{S} =$ MRS (memory random selection), it is a subsample of next memory $\mathbf{M}_{t+1}$, thereby depends the next memory $\mathbf{M}_{t+1}$, see thin dashed (gray) line in Figure 4.3. But when $\mathcal{S} =$ TS (truncate selection), RS (random selection) or ID (identity selection), it is a direct subsample of $\mathbf{X}_t \cup \mathbf{M}_t$, therefore does not depend on $\mathbf{M}_{t+1}$, see thin (gray) line in Figure 4.3. Consequently, for each $\mathbf{n} \in S^{N_b}$,

$$\mathbf{P}\big[\mathcal{N}_t^b = \mathbf{n} \;\big|\; \mathbf{X}_t, \mathbf{\Pi}_t, \mathbf{M}_t, \mathbf{M}_{t+1}, \mathcal{H}_{t-1}\big] = \mathbf{P}\big[\mathcal{N}_t^b = \mathbf{n} \;\big|\; \mathbf{X}_t, \mathbf{M}_t\big]$$

holds for cases $\mathcal{S} =$ ID, TS, or RS, but does not hold for cases $\mathcal{S} =$ MID or MRS.

Similar to $\big(\mathbf{\Pi}_t\big)_{t=0,1,2,\ldots}$, the state space of marginal process $\big(\mathbf{W}_t\big)_{t=0,1,2,\ldots}$ is also the continuous space $\mathbb{P} = \mathbb{P}_{ce}$. In each iteration $t \in \mathbb{N}$, after the subsample $\mathcal{N}_t^b$ is selected, we shall learn an empirical model $\mathbf{W}_t$ from $\mathcal{N}_t^b$ by the fixed learning rule $\mathcal{L}$. Like $\mathbf{\Pi}_t$, each $\mathbf{W}_t = (\mathbf{W}_t(1), \ldots, \mathbf{W}_t(L))$ with $\mathbf{W}_t(i) = \Big(\mathbf{W}_t(a; i)\Big)_{a \in \mathcal{A}}$. When $\mathcal{L} =$ UL (uniform

learning), each $\mathbf{W}_t(a;i)$ is set to be the relative frequency of item a at the i-th position of the subsample $\mathcal{N}_t^b$, see (4.18). And when $\mathcal{L} = \mathrm{WL}$ (weighted learning), $\mathbf{W}_t(a;i)$ will be the weighted frequency of item a at the i-th position of $\mathcal{N}_t^b$, see (4.19). Obviously, $\mathcal{N}_t^b$ dominates $\mathbf{W}_t$, and for each $\mathbf{w} \in \mathbb{P}$,

$$\mathbf{P}\big[\mathbf{W}_t = \mathbf{w} \mid \mathbf{X}_t, \mathbf{\Pi}_t, \mathbf{M}_t, \mathcal{N}_t^b, \mathcal{H}_{t-1}\big] = \mathbf{P}\big[\mathbf{W}_t = \mathbf{w} \mid \mathcal{N}_t^b\big],$$

for each $t \in \mathbb{N}$.

After $\mathbf{W}_t$ is learned, the next model $\mathbf{\Pi}_{t+1}$ is set to be a convex combination of the present mode $\mathbf{\Pi}_t$ and $\mathbf{W}_t$ as in the basic recursion (4.13). In details,

$$\mathbf{\Pi}_{t+1}(a;i) = (1-\rho_{t+1})\mathbf{\Pi}_t(a;i) + \rho_{t+1}\mathbf{W}_t(a;i) \tag{4.21}$$

for all $i = 1, \ldots, L$ and $a \in \mathcal{A}$. Hence, for each $\mathbf{p} \in \mathbb{P}$,

$$\begin{aligned}\mathbf{P}\big[\mathbf{\Pi}_{t+1} = \mathbf{p} \mid \mathbf{X}_t, &\mathbf{\Pi}_t, \mathbf{M}_t, \mathcal{N}_t^b, \mathbf{W}_t, \mathcal{H}_{t-1}\big] \\ &= \mathbf{P}\big[\mathbf{\Pi}_{t+1} = \mathbf{p} \mid \mathbf{\Pi}_t, \mathbf{W}_t\big] = \begin{cases} 1 & \text{if (4.21) holds,} \\ 0 & \text{otherwise,} \end{cases}\end{aligned}$$

for $t \in \mathbb{N}$.

The joint process actually forms a Markov chain. By the above discussion, it is not difficult to see that

$$\begin{aligned}\mathbf{P}\Big[\big(\mathbf{\Pi}_t; \mathbf{X}_t; \mathbf{M}_t; \mathcal{N}_t^b; \mathbf{W}_t\big) = (\mathbf{p}; \mathbf{x}; \mathbf{m}; \mathbf{n}; \mathbf{w}) \;\Big|\; \mathcal{H}_{t-1}\Big] \\ = \mathbf{P}\Big[\big(\mathbf{\Pi}_t; \mathbf{X}_t; \mathbf{M}_t; \mathcal{N}_t^b; \mathbf{W}_t\big) = (\mathbf{p}; \mathbf{x}; \mathbf{m}; \mathbf{n}; \mathbf{w}) \;\Big|\; \big(\mathbf{\Pi}_{t-1}; \mathbf{M}_{t-1}\big)\Big],\end{aligned} \tag{4.22}$$

for $\mathbf{p} \in \mathbb{P}$, $\mathbf{x} \in S^N$, $\mathbf{m} \in S^M$, $\mathbf{n} \in S^{N_b}$ and $\mathbf{w} \in \mathbb{P}$ are constants. In the future, we shall refer to (4.22) as 'Markov property'.

In the next two Chapters, we shall give a more detailed analysis on the joint process. We will see that the basic recursion (4.21), Markov property (4.22), memory update rule $\mathcal{M}$ and subsample selection rule $\mathcal{S}$ take pivotal roles in the analysis.

5 Analysis of model-based search I: reachability and a crude runtime analysis

In Chapter 4, we proposed a unified framework for MBS algorithms, see Figure 4.2. The framework covers the essential features of these MBS algorithms used in practice, i.e. each MBS algorithm may correspond to the framework equipped with a particular strategy where a strategy for the framework is a combination of a sequence of feasibility distributions, a memory update rule, a subsample selection rule, a learning rule and possibly a distribution update rule. With a fixed strategy, the framework finally results in a joint Markov chain $\left(\mathbf{\Pi}_t; \mathbf{X}_t; \mathbf{M}_t; \mathcal{N}_t^b; \mathbf{W}_t\right)_{t=0,1,2,\ldots}$, see Section 4.4. From now on, we shall concentrate on this Markov chain. We will show a detailed mathematical analysis for its asymptotic properties. The results shall carry over to these algorithms which are covered in the framework.

We will focus on four issues in the analysis: conditions for guaranteeing to reach an optimal solution in finitely many iterations (finite reachability); conditions for reaching an optimal solution in a polynomial runtime (runtime analysis); convergence property of marginal process $\left(\mathbf{X}_t\right)_{t=0,1,2,\ldots}$ (absorption of solutions); and convergence property of marginal process $\left(\mathbf{\Pi}_t\right)_{t=0,1,2,\ldots}$ (absorption of models). However, in the present Chapter, we will concentrate only on finite reachability and runtime analysis. The absorption of solutions and models will be inspected in Chapter 6.

Conditions guaranteeing to find an optimal solution in finitely many iterations are of great interests in the theoretical analysis of an optimization algorithm. For some particular MBS algorithms, there are some conditions available in the literature. In [Gut00] and [Gut03], W. J. Gutjahr inspected the conditions on a particular ACO algorithm called GBAS (graph-based ant system). The algorithm learns the empirical distribution only with the best solution found so far, and uses a constant learning rate $\rho_t \equiv \rho \in (0, 1)$. In [Gut00], W. J. Gutjahr showed that when $|S^*| = 1$ i.e. there is a unique optimal solution, the probability for GBAS to reach an optimal solution in finitely many iterations approaches 1 as the sample size $N \to \infty$ or the constant learning rate $\rho \to 0$. In [Gut03], he extended the result proposed in [Gut00] by removing the restriction $|S^*| = 1$, i.e. he showed that his result in [Gut00] still holds even when $|S^*| > 1$. In [CJK07], A. Costa et al inspected the conditions for a generalized cross entropy algorithm on unconstrained problems, i.e. CE/tdsp (cross entropy algorithm equipped with a sequence of smooth parameters $\{\rho_t\}_{t\in\mathbb{N}}$). They found conditions which imply CE/tdsp reach an optimal solution in finitely many iterations with probability 1. In [WK14b], we extended the conditions in [CJK07] to a more general CE i.e. CE/as (ant-like CE, our framework without memory). In Section 5.3, we continue [Gut00], [Gut03], [CJK07] and [WK14b]. We show that those conditions in the literature still hold in the unified framework, see

Theorem 5.7. The Theorem does not assume a particular algorithm, and does not impose a particular structure of the underlying CO instances. Therefore, it should apply to all MBS algorithms which are covered in the framework and to arbitrary CO instance. In particular, it shows that in the case of $\rho_t \equiv \rho > 0$ for a constant ρ, we may never see an optimal solution. Of cause, here we avoid the trivial case that $S = S^*$.

Note that $\rho_t \equiv \rho > 0$ is a common setting in practice, see Table 4.1. To show more insight into this case, two runtime results will be proposed in Section 5.4. Recently, runtime analysis of heuristic algorithms becomes a very popular research field, see e.g. [NW06], [DNSW07], [DJ07], [NW09], [Gut07], [Gut08], [CTCY10] and [WK14b]. In runtime analysis, we consider conditions which make the algorithm reach an optimal solution efficiently (i.e. in a polynomial runtime) with a large probability. Here, the runtime is a rudimentary copy of the computation complexity [AB09] in theoretical computer science. Due to the famous No Free Lunch Theorem [WM97], we need to assume a particular test problem in such an analysis. In [NW06], [DNSW07], [DJ07] and [NW09], the authors proposed some runtime results for a so-called 1-ANT ACO algorithm (a simple $\mathcal{MMAS}$ with sample size $N = 1$ and update models only when new improvement occurs) on some simple test problems e.g. OneMax and LeadingOne. In [Gut07], W. J. Gutjahr summarized the commonly used techniques in runtime analysis. In [Gut08], W. J. Gutjahr proposed a polynomially expected runtime for $\mathcal{MMAS}$ on the OneMax problem. In [CTCY10], Chen et al showed a polynomial runtime for a UMDA (univariate marginal distribution algorithm with truncate selection) on the LeadingOne problem. In [WK14b], we showed a polynomial runtime for CE on the LeadingOne problem. Note that the UDMA inspected in [CTCY10] is actually a particular CE with a constant smooth parameter $\rho = 1$. The runtime result in [WK14b] therefore extends the result in [CTCY10]. Note also that both 1-ANT and $\mathcal{MMAS}$ employ restricted models and anew models only by best solution found so far. However, CE does not use restricted models, and update models by present solutions (elite solutions). Thereby, our former result actually complements the results in [NW06], [DNSW07], [DJ07], [NW09] and [Gut08]. In Section 5.3, we shall collect our former runtime result in Theorem 5.8. To further understand the role of subsample selection in efficiently finding an optimal solution, we propose a new runtime result in Theorem 5.9.

The whole Chapter is arranged as: Section 5.1 formally defines the finite reachability and runtime, and collects some necessary assumptions which are generally fulfilled in practice; Section 5.2 collects some helpful properties of the underlying process; Section 5.3 concentrates on finite reachability; Section 5.4 concentrates the runtime analysis.

5.1 Definitions and assumptions

Recall that in the framework the underlying CO instance $(S, f, 0)$ is string encoded, i.e. $S \subseteq \mathcal{A}^L$ for a finite alphabet $\mathcal{A}$ and a constant $L \in \mathbb{N}$. The model family is specified to be $\mathbb{P} = \mathbb{P}_{ce}$, for a definition see (3.2) on p. 17.

5.1.1 Definitions

Let $\tau \in \mathbb{N}$ be a random variable denoting the first hitting time (iteration) for an optimal solution i.e.

$$\tau := \min\big\{\{t \in \mathbb{N} \mid \mathbf{X}_t \cap S^* \neq \emptyset\} \cup \{\infty\}\big\} \tag{5.1}$$

where S^* is the collection of optimal solutions. Then if $\mathbf{P}\big[\tau < \infty\big] = 1$, we say that (almost surely) *finite reachability* holds. Obviously, finite reachability concerns the effectiveness of the framework. Therefore, conditions which make finite reachability hold in the framework should be of great importance. Section 5.3 shall show conditions which make finite reachability hold in the framework. The conditions do not impose any restrictions on the underlying CO instance, thereby apply to all combinatorial optimization problems.

In the literature, see e.g. [NW06], [DNSW07], [DJ07], [NW09], [Gut07], [Gut08], [CTCY10] and [WK14b], *runtime* of an algorithm is generally defined as the total number of feasible solutions evaluated before reaching an optimal solution. Note that this is a rudimentary copy of the computational complexity [AB09]. In the framework, the runtime is obviously $N \cdot \tau$. To express the efficiency of an algorithm, runtime is often formulated as a function in problem size. In the string encoding, the *problem size* can be described by the encoded solutions length. Formally, we say that an algorithm has a polynomial runtime with degree $n \in \mathbb{N}$ on a *problem*[1] $\mathcal{P}$ if for each instance $(S', f', 0) \in \mathcal{P}$ with the encoded solutions length $L_{(S',f',0)} \geq L_{threshold} \in \mathbb{N}$, the runtime of the algorithm is bounded above by a polynomial function $c \cdot \big(L_{(S',f',0)}\big)^n$, where $L_{threshold}$ and $c \in \mathbb{R}_+$ are constants. By the standard notation in complexity, we can denote this polynomial runtime as

$$O\left(\left(L_{(S',f',0)}\right)^n\right)$$

where $L_{(S',f',0)}$ is solution length of an arbitrary instance in problem $\mathcal{P}$.[2] Algorithms with polynomial runtime are considered as efficient, and with low degree polynomial are practically preferred. Comparatively, an algorithm is inefficient on a problem $\mathcal{P}$ if the runtime is *exponential* i.e. for each $(S', f', 0) \in \mathcal{P}$ the runtime is asymptotically bounded below by an exponential function $c^{L_{(S',f',0)}}$ for some constant $c > 1$. In runtime analysis, we inspect conditions which imply (low degree) polynomial runtime with a high probability. Section 5.4 will present conditions which imply a polynomial runtime on the two testing problem i.e. OneMax and LeadingOne, with an overwhelming probability.

[1] Recall that a problem is a class of instance.

[2] See A. 5 in the Appendix for the big O notation and other notations used in runtime analysis.

5.1.2 General assumptions

In the sequel, we assume that the starting model $\mathbf{\Pi}_0$ satisfies that any item from $\mathcal{A}$ has a positive probability at any position i :

$$\mathbf{\Pi}_0(a;i) > 0 \qquad \text{for all } a \in \mathcal{A}, i = 1, \ldots, L. \tag{5.2}$$

Note that in practice, $\mathbf{\Pi}_0$ is generally taken as a uniform distribution over $\mathcal{A}^L$ which meets the assumption (5.2).

Without loss in generality, we may also assume that there are at least two solutions

$$s = (s_1, \ldots, s_L), s' = (s'_1, \ldots, s'_L) \;\text{ with }\; s_1 \neq s'_1. \tag{5.3}$$

Otherwise, all solutions would start with the same symbol, which could then be dropped from the encoding of the solutions.

Note that for a feasible solution $s = (s_1, \ldots, s_L) \in S$, the sampling probability (4.5)

$$Q\big(s;\mathbf{\Pi}\big) = \prod_{i=0}^{L-1} Q\big(s_{i+1}; s_{|i}, i+1, \mathbf{\Pi}\big) \quad \text{with each}$$
$$Q(s_{i+1}; s_{|i}, i+1, \mathbf{\Pi}) = \frac{\mathbf{\Pi}(s_{i+1}; i+1) C_i(s_{|i}; s_{i+1})}{\sum_{a' \in C_i(s_{|i})} \mathbf{\Pi}(a'; i+1) C_i(s_{|i}; a')},$$

may be zero even in the case of an uniform $\mathbf{\Pi} \in \mathbb{P}$, where recall that $s_{|i} = (s_1, \ldots, s_i)$ and $s_{|0} = \diamond$. Therefore, the random solution generation algorithm in Figure 4.1 may run a risk that an optimal solution can not be sampled under any model $\mathbf{\Pi} \in \mathbb{P}$. To save this, we assume that the constructed feasibility distributions $\{C_i\}_{i=0}^{L-1}$ are *compatible* to the optimal solutions, i.e.

$$\prod_{i=0}^{L-1} C_i\big(s^*_{|i}; s^*(i+1)\big) > 0 \quad \text{for all } s^* \in S^*. \tag{5.4}$$

And to avoid trivial cases, we assume further that

$$S \neq S^* \quad \text{and} \quad \prod_{i=0}^{L-1} C_i\big(s_{|i}; s(i+1)\big) > 0 \;\text{ for some } s \in S - S^*. \tag{5.5}$$

Otherwise, the sampling would always produce optimal solutions.

Note that in MBS algorithms, we often need to compare objective values of two different solutions. But, it may occur that two different solutions have the same objective value. To deal with this embarrassing situation, we assume that

$$\text{for each iteration } t \in \mathbb{N} \quad \text{if } f(\mathbf{X}_t^{(n)}) = f(\mathbf{X}_t^{(m)}) \text{ and } n > m \tag{5.6}$$

then we think $\mathbf{X}_t^{(n)}$ is *better* than $\mathbf{X}_t^{(m)}$, and

$$\text{for each pair of iterations } t \in \mathbb{N} \text{ and } t' \in \mathbb{N} \quad \text{if } f(X_t^{(m)}) = f(\mathbf{X}_{t'}^{(n)}) \text{ and } t < t' \tag{5.7}$$

then we think $\mathbf{X}_{t'}^{(n)}$ is *better* than $\mathbf{X}_{t}^{(m)}$. Note that assumptions (5.6) and (5.7) actually assign the time order to solutions of the same objective value i.e. we think that new solutions are better than old solutions if they have the same objective values. The assumptions will be employed only when we need to sort solutions in a collection.

Finally, we give assumptions on the empirical distribution $\mathbf{W}_t$ learned by rule $\mathcal{L}$. We assume that:

i) for all $a \in \mathcal{A}, i \in \{1, \ldots, L\}, t \in \mathbb{N}$,

$$\forall s = (s_1, \ldots, s_L) \in \mathcal{N}_t^b \quad s_i \neq a \;\Rightarrow\; \mathbf{W}_t(a; i) = 0, \tag{5.8}$$

$$\forall s = (s_1, \ldots, s_L) \in \mathcal{N}_t^b \quad s_i = a \;\Rightarrow\; \mathbf{W}_t(a; i) = 1; \tag{5.9}$$

ii) there exists a constants $\alpha \in (0, 1)$ such that for all $a \in \mathcal{A}, i \in \{1, \ldots, L\}, t \in \mathbb{N}$,

$$\mathbf{W}_t(a; i) > 0 \;\Rightarrow\; \mathbf{W}_t(a; i) > \alpha, \tag{5.10}$$

$$\mathbf{W}_t(a; i) < 1 \;\Rightarrow\; \mathbf{W}_t(a; i) < 1 - \alpha. \tag{5.11}$$

Note that these assumptions are fulfilled in the practically used learning rules. Recall that practically used learning rules are of two types, namely UL (uniform learning, see (4.18) on p. 60) and WL (weighted learning, see (4.19)). In UL, $\mathbf{W}_t(a; i)$ is the relative frequency of letter a at position i in the selected subsample $\mathcal{N}_t^b$ i.e.

$$\mathbf{W}_t(a; i) := \frac{\sum_{s \in \mathcal{N}_t^b} \mathbb{1}_{\{a\}}(s_i)}{|\mathcal{N}_t^b|} \text{ for all } a \in \mathcal{A} \text{ and } i = 1, \ldots, L.$$

Obviously, (5.8) and (5.9) are fulfilled in this case. In WL, each solution in $\mathcal{N}_t^b$ is allocated a *positive* weight according to its cost value. And then $\mathbf{W}_t(a; i)$ calculates the weighted relative frequency of a at position i in $\mathcal{N}_t^b$ i.e.

$$\mathbf{W}_t(a; i) := \frac{\sum_{s \in \mathcal{N}_t^b} \mathbb{1}_{\{a\}}(s_i) g(s)}{\sum_{s \in \mathcal{N}_t^b} g(s)} \text{ for all } a \in \mathcal{A} \text{ and } i = 1, \ldots, L,$$

where g is a positive weight function. Also (5.8) and (5.9) are fulfilled. Both UL and WL are time-independent learning, i.e. $\mathbf{W}_{t_1} = \mathbf{W}_{t_2}$ if $\mathcal{N}_{t_1}^b = \mathcal{N}_{t_2}^b$, for all $t_1 \in \mathbb{N}$, $t_2 \in \mathbb{N}$. Since the state space of $\left(\mathcal{N}_t^b\right)_{t \in \mathbb{N}}$ is S^{N_b} and $|S^{N_b}| < \infty$, so

$$\alpha' := \min\left\{\mathbf{W}_t(a; i) > 0 \;\middle|\; a \in \mathcal{A}, i = 1, \ldots, L, t \in \mathbb{N}\right\} > 0,$$
$$\beta' := \max\left\{\mathbf{W}_t(a; i) < 1 \;\middle|\; a \in \mathcal{A}, i = 1, \ldots, L, t \in \mathbb{N}\right\} < 1.$$

Then obviously $\alpha := \min\{\alpha', 1 - \beta'\}$ fulfills (5.10) and (5.11).

For time-dependent learning (4.20), assumptions (5.8) and (5.9) generally still holds, but (5.10) and (5.11) may not hold. However, most of the statements in Chapter 5 and Chapter 6 will still hold, if the allocated learning weights in time-dependent learning are not changed too dramatically as iteration varies. To simplify the discussion, we will stick to the case of uniform learning or weighted learning in the sequel.

5.2 Some basic properties

As a start of the analysis, we now collect some useful properties about the models $(\mathbf{\Pi}_t)_{t\in\mathbb{N}}$ and sampling probabilities (see (4.5) on p. 52), which will be frequently used in the present Chapter and Chapter 6.

5.2.1 On basic recursion

Recall that basic recursion (4.21)

$$\mathbf{\Pi}_{t+1} = (1-\rho_{t+1})\mathbf{\Pi}_t + \rho_{t+1}\mathbf{W}_t$$

is the employed distribution update rule in the framework, where $\mathbf{\Pi}_t \in \mathbb{P} = \mathbb{P}_{ce}$ is the present model, $\mathbf{W}_t \in \mathbb{P} = \mathbb{P}_{ce}$ is the model learned from a selected subsample $\mathcal{N}_t^b$, $\rho_{t+1} > 0$ is the learning rate for time $t+1$, and $\mathbf{\Pi}_{t+1} \in \mathbb{P} = \mathbb{P}_{ce}$ is the model for next round, for $t = 0, 1, 2, \ldots$. To facilitate our analysis, we now collect some useful statements closely related to this basic recursion in the following Lemma. Here, we use a convention that empty product equals 1, i.e. $\prod_{m=t+1}^{t} \cdots = 1$.

Lemma 5.1. *Let $0 < r_t < 1$ for $t = 1, 2, \ldots$.*

a) $\displaystyle\sum_{t=1}^{\infty} r_t = \infty \iff \prod_{t=1}^{\infty}(1-r_t) = 0 \iff \prod_{t=1}^{\infty}(1-cr_t) = 0$ *for any* $0 < c < 1$.

b) $\displaystyle\sum_{m=1}^{t} r_m \prod_{i=m+1}^{t}(1-r_i) = 1 - \prod_{m=1}^{t}(1-r_m)$ *for any* $t \geq 1$.

c) *For a given sequence $w_t \in [0,1], t = 0, 1, \ldots$ and $q_0 \in (0,1)$, the recursion*

$$q_{t+1} = (1-r_{t+1})q_t + r_{t+1}w_t, \quad t \geq 0, \tag{5.12}$$

has the unique solution

$$q_t = q_0 \prod_{m=1}^{t}(1-r_m) + \sum_{m=1}^{t} r_m w_{m-1} \prod_{i=m+1}^{t}(1-r_i) \tag{5.13}$$

with

$$0 < q_0 \prod_{m=1}^{t}(1-r_m) \leq q_t \leq 1 - (1-q_0)\prod_{m=1}^{t}(1-r_m) < 1, \quad t \geq 0. \tag{5.14}$$

If, in particular, $w_m \equiv w \in [0,1]$ for $m = 0, \ldots, t-1$ then

$$q_t = w - (w-q_0)\cdot\prod_{m=1}^{t}(1-r_m). \tag{5.15}$$

And furthermore if we allow some $r_t = 1$, then b), (5.13) *and* (5.15) *still hold, and* (5.14) *becomes*

$$0 \leq q_0 \prod_{m=1}^{t}(1-r_m) \;\leq\; q_t \leq\; 1-(1-q_0)\prod_{m=1}^{t}(1-r_m) \leq 1, \quad t \geq 0.$$

Proof. (see also [WK14b] and [WK14a])
a) The first part is a standard result of infinite products, see e.g. [Kno90], the second follows as

$$\sum_{i=0}^{\infty} r_i = \infty \iff \sum_{i=0}^{\infty} c r_i = \infty.$$

b) Note that $r_i = 1-(1-r_i)$. Then

$$\begin{aligned}
\sum_{m=1}^{t} r_m \prod_{i=m+1}^{t}(1-r_i) &= \sum_{m=1}^{t}\big[1-(1-r_m)\big]\prod_{i=m+1}^{t}(1-r_i) \\
&= \sum_{m=1}^{t}\prod_{i=m+1}^{t}(1-r_i) - \sum_{m=1}^{t}\prod_{i=m}^{t}(1-r_i) \\
&= 1+\sum_{m=1}^{t-1}\prod_{i=m+1}^{t}(1-r_i) - \sum_{m=2}^{t}\prod_{i=m}^{t}(1-r_i) - \prod_{m=1}^{t}(1-r_m) \\
&= 1-\prod_{m=1}^{t}(1-r_m).
\end{aligned}$$

c) (5.13) can be proven using induction on t. (5.15) follows from (5.13) using b). Then (5.14) follows immediately from the facts that

$$0 \;\leq\; \sum_{m=1}^{t} r_m w_{m-1} \prod_{i=m+1}^{t}(1-r_i) \;\leq\; \sum_{m=1}^{t} r_m \prod_{i=m+1}^{t}(1-r_i) = 1-\prod_{m=1}^{t}(1-r_m).$$

□

Applying Lemma 5.1 c) to the basic recursion (4.21), we immediately get that

Lemma 5.2. *If the learning rate $\rho_t \in (0,1]$ for each $t = 1, 2, \ldots$, then the following hold for all $t, k \geq 0$, $a \in \mathcal{A}$ and $i = 1, \ldots, L$:*

a) $\mathbf{\Pi}_t(a;i)\prod_{m=1}^{k}(1-\rho_{t+m}) \leq \mathbf{\Pi}_{t+k}(a;i) \leq 1-\big(1-\mathbf{\Pi}_t(a;i)\big)\prod_{m=1}^{k}(1-\rho_{t+m})$;

b) $\mathbf{W}_t(a;i) = \cdots = \mathbf{W}_{t+k-1}(a;i) = 0 \;\Rightarrow\; \mathbf{\Pi}_{t+k}(a;i) = \mathbf{\Pi}_t(a;i)\prod_{m=1}^{k}(1-\rho_{t+m})$, *and if we assume further that $\rho_t \in (0,1)$ for all $t \in \mathbb{N}$, then the inverse implication also holds;*

c) *for all* $m = 0, \ldots, k-1$

$$\mathbf{W}_{t+m}(a;i) \equiv 1 \;\Rightarrow\; \mathbf{\Pi}_{t+k}(a;i) = 1 - \big(1 - \mathbf{\Pi}_t(a;i)\big) \prod_{m=1}^{k} (1-\rho_{t+m}),$$

and if we assume further that $\rho_t \in (0,1)$ *for all* $t \in \mathbb{N}$, *then the inverse implication also holds;*

d) *if* $\rho_m \in (0,1)$ *for all* $m \le t$, $0 < \mathbf{\Pi}_t(a;i) < 1$;

e) *if* $\prod_{t=1}^{\infty}(1-\rho_t) = 0$ *and* $\lim_{t\to\infty} \mathbf{W}_t(a;i) = w$, *then* $\lim_{t\to\infty} \mathbf{\Pi}_t(a;i) = w$.

Proof. Statements a)-d) follows immediately from c) of Lemma 5.1 with observation to the assumption (5.2). Now, we prove statement e).

Assume that $\prod_{t=1}^{\infty}(1-\rho_t) = 0$ and $\mathbf{W}_t(a;i) \to w$ as $t \to \infty$. Let $\epsilon > 0$ be an arbitrarily fixed constant. Then there exists an $T \in \mathbb{N}$ such that

$$w + \epsilon \ge \mathbf{W}_t(a;i) \ge w - \epsilon \quad \text{for all } t \ge T.$$

Then by (5.13) and b) of Lemma 5.1, we have that

$$\begin{aligned}
(w+\epsilon) - \big((w+\epsilon) - 1\big) \prod_{m=T+1}^{t} (1-\rho_m) \ge \\
(w+\epsilon) - \big((w+\epsilon) - \mathbf{\Pi}_T(a;i)\big) \prod_{m=T+1}^{t} (1-\rho_m) \ge \\
\mathbf{\Pi}_t(a;i) \ge (w-\epsilon) - \big((w-\epsilon) - \mathbf{\Pi}_T(a;i)\big) \prod_{m=T+1}^{t} (1-\rho_m) \\
\ge (w-\epsilon) - (w-\epsilon) \prod_{m=T+1}^{t} (1-\rho_m)
\end{aligned}$$

for all $t \ge T+1$. Since $\prod_{m=1}^{\infty}(1-\rho_m) = 0$, $\prod_{m=T+1}^{\infty}(1-\rho_m) = 0$ for any fixed $T \in \mathbb{N}$. Consequently, we have

$$w + \epsilon \ge \limsup_{t\to\infty} \mathbf{\Pi}_t(a;i) \ge \liminf_{t\to\infty} \mathbf{\Pi}_t(a;i) \ge w - \epsilon.$$

Note that ϵ is arbitrarily fixed, this results in

$$w = \limsup_{t\to\infty} \mathbf{\Pi}_t(a;i) = \liminf_{t\to\infty} \mathbf{\Pi}_t(a;i) = \lim_{t\to\infty} \mathbf{\Pi}_t(a;i).$$

□

Lemma 5.2 e) actually means that the limiting behavior of the models is dominated by the limiting behavior of the learned empirical distributions. Hence, in the sequel

when we want to derive a limit of the models, we often derive a limit for the empirical distributions in advance.

For a constructed feasibility distributions $\{C_i\}_{i=0}^{L-1}$ and a feasible solution $s = (s_1, \ldots, s_L) \in S$, we say that s is *compatible to* or *feasible under* $\{C_i\}_{i=0}^{L-1}$ if and only if

$$\prod_{i=0}^{L-1} C_i\big(s_{|i}; s_{i+1}\big) > 0.$$

As a result of Lemma 5.2 d), we have immediately that

Lemma 5.3. *Let $\{C_i\}_{i=0}^{L-1}$ be the feasibility distributions and $\rho_t \in (0,1)$ for all $t \in \mathbb{N}$, then for any $s \in S$ which is compatible to $\{C_i\}_{i=0}^{L-1}$, the probability $Q(s; \mathbf{\Pi}_t) > 0$ for all $t \geq 0$ where Q is defined in (4.5), i.e.*

$$Q(s; \mathbf{\Pi}_t) = \prod_{i=0}^{L-1} Q\big(s_{i+1}; s_{|i}, i+1, \mathbf{\Pi}_t\big)$$

and

$$Q\big(s_{i+1}; s_{|i}, i+1, \mathbf{\Pi}_t\big) = \frac{C_i(s_{|i}; s_{i+1})\mathbf{\Pi}_t(s_{i+1}; i+1)}{\sum_{a \in C_i(s_{|i})} C_i(s_{|i}; a)\mathbf{\Pi}_t(a; i+1)}. \tag{5.16}$$

In particular, for each $s^* \in S^*$, $Q(s^*; \mathbf{\Pi}_t) > 0$ for all $t \geq 0$.

Note that a feasible solution can be sampled only if it is compatible to the constructed feasibility distributions. Due to this fact, in the sequel when we refer to feasible solutions, we mean those which are feasible under (compatible to) the feasibility distributions. Note also that with assumption (5.4), optimal solutions are always feasible under the feasibility distributions. Hence, Lemma 5.3 shows that in each iteration optimal solutions can always have a positive chance to be sampled provided $\rho_t \in (0,1)$ for all $t \in \mathbb{N}$. But in the case $\rho_t = 1$ for some $t \in \mathbb{N}$, this is not guaranteed.

5.2.2 A surrogate probability of $Q(\cdot; y, i, \mathbf{\Pi})$ and its properties

To theoretically study the framework, we need to study the letter selection probability $Q(\cdot; y, i, \mathbf{\Pi})$ (defined in (4.4), see also (5.16)) more closely.

For the unconstrained non-greedy feasibility construction (see (4.6)), $Q(a; y, i, \mathbf{\Pi}) = \mathbf{\Pi}(a; i)$ holds for all $a \in \mathcal{A}$, $\mathbf{\Pi} \in \mathbb{P}$ and $i = 1, \ldots, L$. Therefore, in this case Lemma (5.1) c) directly applies to $Q(a; y, i, \mathbf{\Pi}_t)$ for all $a \in \mathcal{A}$, $y \in R_{i-1}$, $t \geq 0$ and $i = 1, \ldots, L$. In general, this is not possible as $Q(a; y, i, \mathbf{\Pi}_t)$ does not fulfill the recursion (5.12). But we can find a *surrogate probability* $Q'(a; y, i, \mathbf{\Pi}_t)$ for $Q(a; y, i, \mathbf{\Pi}_t)$, for which a similar recursion holds under certain conditions.

The surrogate probability $Q'(a; y, i, \mathbf{\Pi}_t)$ using here is also the one defined in (4.7), i.e. the selection probability for non-greedy feasibility construction. For a feasible partial solution $y \in R_{i-1}$, a letter $a \in \mathcal{A}$, a position $i = 1, \ldots, L$, and a model $\mathbf{\Pi} \in \mathbb{P}$, we set

$$Q'(a; y, i, \mathbf{\Pi}) := \frac{\mathbb{1}_{C_{i-1}(y)}(a)\mathbf{\Pi}(a; i)}{\sum_{b \in \mathcal{A}} \mathbb{1}_{C_{i-1}(y)}(b)\mathbf{\Pi}(b; i)} \tag{5.17}$$

where we again use the convention that $\frac{0}{0} = 0$. Obviously, for an unconstrained non-greedy feasibility construction, we have $Q(a; y, i, \mathbf{\Pi}) = Q'(a; y, i, \mathbf{\Pi}) = \mathbf{\Pi}(a; i)$, and for a non-greedy feasibility construction, we have $Q(a; y, i, \mathbf{\Pi}) = Q'(a; y, i, \mathbf{\Pi})$.

In general, although $Q(a; y, i, \mathbf{\Pi}) \neq Q'(a; y, i, \mathbf{\Pi})$, we can employ this surrogate probability to bound Q. To do this, we need bounds for the constructed feasibility distributions. Let

$$\eta := \max \{C_i(y; a) | y \in R_i, a \in \mathcal{A}, i = 0, 1, 2, \ldots, L-1\} \tag{5.18}$$

and

$$\lambda := \min \{C_i(y; a) > 0 | y \in R_i, a \in \mathcal{A}, i = 0, 1, 2, \ldots, L-1\}. \tag{5.19}$$

Obviously, $1 \geq \eta \geq \lambda > 0$. Now, we define two bounding functions $\hbar(x)$ and $\ell(x)$ for $x \in [0, 1]$ as

$$\hbar(x) := \frac{\eta x}{\lambda + (\eta - \lambda)x} \text{ and } \ell(x) := \frac{\lambda x}{\eta - (\eta - \lambda)x}. \tag{5.20}$$

The following Lemma collects some properties of these two bounding functions.

Lemma 5.4. ***a)*** *$\ell(x) \leq x \leq \hbar(x)$, $\hbar(x) = 1 - \ell(1 - x)$ and $\ell(x) = 1 - \hbar(1 - x)$.*

b) *$\hbar$ and ℓ are both continuous and strictly increasing with $\ell(0) = \hbar(0) = 0$, $\ell(1) = \hbar(1) = 1$.*

c) *Suppose $\{x_n\}_{n \in \mathbb{N}}$ is a convergent sequence in $[0, 1]$ then for any constant $c \in (0, 1]$, we have $\sum_n x_n < \infty \iff \sum_n \hbar(cx_n) < \infty \iff \sum_n \ell(cx_n) < \infty$.*

The proof of Lemma 5.4 can be found in [WK14b]. With the help of Q', $\hbar$ and ℓ, we can bound Q as

Lemma 5.5. *Let $i \in \{0, 1, 2, \ldots, L-1\}, y \in R_i$, and $a \in C_i(y)$. Then the following holds.*

a) *$\ell\big(Q'(a; y, i+1, \mathbf{\Pi}_t)\big) \leq Q(a; y, i+1, \mathbf{\Pi}_t) \leq \hbar\big(Q'(a; y, i+1, \mathbf{\Pi}_t)\big)$ for all $t \in \mathbb{N}$.*

b) *Assume $\rho_t \in (0, 1)$ for all $t \in \mathbb{N}$. If $|C_i(y)| = 1$ then $Q(a; y, i+1, \mathbf{\Pi}_t) = Q'(a; y, i+1, \mathbf{\Pi}_t) = 1$. If $|C_i(y)| > 1$, then $0 < Q'(a; y, i+1, \mathbf{\Pi}_t) < 1$ and $0 < Q(a; y, i+1, \mathbf{\Pi}_t) < 1$ for any $t \geq 0$.*

Proof. **a)**

$$\begin{aligned}
Q(a;y,i+1,\mathbf{\Pi}_t) &= \frac{\mathbf{\Pi}_t(a;i+1)C_i(y;a)}{\sum_{a'\in\mathcal{A}}\mathbf{\Pi}_t(a';i+1)C_i(y;a')} \\
&= \frac{C_i(y;a)\mathbf{\Pi}_t(a;i+1)}{C_i(y;a)\mathbf{\Pi}_t(a;i+1)+\sum_{a'\in C_i(y),a\neq a'}\mathbf{\Pi}_t(a';i+1)C_i(y;a')} \\
&\leq \frac{\eta\mathbf{\Pi}_t(a;i+1)}{\eta\mathbf{\Pi}_t(a;i+1)+\sum_{a'\in C_i(y),a\neq a'}\mathbf{\Pi}_t(a';i+1)C_i(y;a')} \\
&\leq \frac{\eta\mathbf{\Pi}_t(a;i+1)}{\eta\mathbf{\Pi}_t(a;i+1)+\sum_{a'\in C_i(y),a\neq a'}\lambda\mathbf{\Pi}_t(a';i+1)} \\
&= \frac{\eta\mathbf{\Pi}_t(a;i+1)}{\eta\mathbf{\Pi}_t(a;i+1)+\lambda\sum_{a'\in C_i(y),a\neq a'}\mathbf{\Pi}_t(a';i+1)} \\
&= \frac{\eta\mathbf{\Pi}_t(a;i+1)}{(\eta-\lambda)\mathbf{\Pi}_t(a;i+1)+\lambda\sum_{a'\in C_i(y)}\mathbf{\Pi}_t(a';i+1)} \\
&= \frac{\eta Q'(a;y,i+1,\mathbf{\Pi}_t)}{(\eta-\lambda)Q'_{\mathbf{\Pi}_t}(a;y,i+1,\mathbf{\Pi}_t)+\lambda} \\
&= \hbar\big(Q'(a;y,i+1,\mathbf{\Pi}_t)\big),
\end{aligned}$$

Similarly, the left-hand inequality of a) is derived.

b) From d) of Lemma 5.2 and (5.17), we have $Q'(a;y,i+1,\mathbf{\Pi}_t) > 0$ for all $t \in \mathbb{N}$. If $|C_i(y)| > 1$, we see that $Q'(a;y,i+1,\mathbf{\Pi}_t) < 1$ for any $a \in C_i(y)$ and $t \in \mathbb{N}$, this implies the conclusion by Lemma 5.4 b) and part a) of this Lemma. □

As we have mentioned above, the surrogate probability fulfills the recursion (5.12) under certain conditions. Now, we give such a condition in (5.23) and show the recursion relation in Lemma 5.6 below. Before this, we need two auxiliary definitions.

For a letter $a \in \mathcal{A}$, a position $i = 1,\ldots,L$ and a partial feasible solution $y \in R_{i-1}$, we define

$$G_{i-1}(y,\mathbf{\Pi}_t) := \sum_{a\in C_{i-1}(y)} \mathbf{\Pi}_t(y;a) \text{ and} \tag{5.21}$$

$$\rho_t^y := \frac{\rho_t}{G_{i-1}(y,\mathbf{\Pi}_t)} \tag{5.22}$$

for $t \geq 1$.

Lemma 5.6. *Assume $\rho_t \in (0,1]$ for all $t \in \mathbb{N}$. Let $i \in \{0,1,2,\ldots,L-1\}$ be fixed and assume*

$$\forall m-1 \geq l \geq 0 \qquad \sum_{a\in C_i(y)} \mathbf{W}_{T+l}(a;i+1) = 1 \tag{5.23}$$

holds for an arbitrary partial solution $y \in R_i$, a random time $T \in \mathbb{N}$ and some constant $m \in \mathbb{N}$. Then the following hold.

a) *For all* $m \geq m' \geq 1$,

$$G_i(y, \mathbf{\Pi}_{T+m'}) = 1 - \big(1 - G_i(y, \mathbf{\Pi}_T)\big) \prod_{l=1}^{m'} (1 - \rho_{T+l}) \quad \textit{and}$$

$$0 < 1 - \prod_{l=1}^{m'} (1 - \rho_{T+l}) \leq G_i(y, \mathbf{\Pi}_{T+m'}) \leq 1.$$

b) *For all* $m \geq l \geq 1$, $\quad 0 < \rho_{T+l} \leq \rho^y_{T+l} \leq 1$. *And furthermore*

$$\rho_{T+l} < 1 \iff \rho^y_{T+l} < 1 \quad \textit{for all } m \geq l \geq 2,$$

and if $\rho_t \in (0,1)$ *for all* $0 < t \leq T$ *then we have*

$$\rho_{T+1} < 1 \iff \rho^y_{T+1} < 1.$$

c) *For all* $a \in C_i(y)$ *and all* $m \geq l \geq 1$ *we have*

$$Q'(a; y, i+1, \mathbf{\Pi}_{T+l}) = (1 - \rho^y_{T+l}) Q'(a; y, i+1, \mathbf{\Pi}_{T+l}) + \rho^y_{T+l} \mathbf{W}_{T+l}(a; i+1). \tag{5.24}$$

Thus the implications of Lemma 5.1 c) hold with $q_t := Q'(a; y, i+1, \mathbf{\Pi}_{T+t})$, $r_t := \rho^y_{T+t}$ *and* $w_t := \mathbf{W}_{T+t}(a; i+1)$.

d) *If* $\mathbf{W}_{T+l}(a; i+1) = w \in [0,1]$ *for all* $l = 0, \ldots, m-1$, *then*

$$Q'(a; y, i+1, \mathbf{\Pi}_{T+m}) = w - \big(w - Q'(a; y, i+1, \mathbf{\Pi}_T)\big) \prod_{l=1}^{m} (1 - \rho^y_{T+l}) \tag{5.25}$$

$$\geq w - \big(w - Q'(a; y, i+1, \mathbf{\Pi}_T)\big) \prod_{l=1}^{m} (1 - \rho_{T+l}) \tag{5.26}$$

Proof of Lemma 5.6. **a)** from the basic recursion (4.21) we have for any $m \geq l \geq 1$

$$\begin{aligned} G_i(y, \mathbf{\Pi}_{T+l}) &= \sum_{a' \in C_i(y)} \mathbf{\Pi}_{T+l}(a'; i+1) \qquad (5.27) \\ &= \sum_{a' \in C_i(y)} \Big((1 - \rho_{T+l}) \mathbf{\Pi}_{T+l-1}(a'; i+1) + \rho_{T+l} \, \mathbf{W}_{T+l-1}(a'; i+1) \Big) \\ &= (1 - \rho_{T+l}) G_i(y, \mathbf{\Pi}_{T+l-1}) + \rho_{T+l}, \end{aligned}$$

as $\sum_{a' \in C_i(y)} \mathbf{W}_{T+l-1}(a'; i+1) = 1$ under assumption (5.23). Hence, $q_t := G_i(y, \mathbf{\Pi}_{T+t})$ fulfills the condition (5.12) of Lemma 5.1 with $w_l \equiv 1$. Now (5.15) shows that

$$G_i(y, \mathbf{\Pi}_{T+m'}) = 1 - \big(1 - G_i(y, \mathbf{\Pi}_T)\big) \prod_{l=1}^{m'} (1 - \rho_{T+l}).$$

Obviously, $G_i(y, \mathbf{\Pi}_T) \in [0,1]$ and

$$1 - \big(1 - G_i(y, \mathbf{\Pi}_T)\big) \prod_{l=1}^{m'} (1 - \rho_{T+l})$$

is increasing as $G_i(y, \mathbf{\Pi}_T) \in [0,1]$, so the remaining holds.

b) From (5.27) we now see $0 < \rho_{T+l} \leq G_i(y, \mathbf{\Pi}_{T+l})$, hence $\rho_t^y \leq 1$ for all $m \geq l \geq 1$ By a) of this Lemma $0 < G_i(y, \mathbf{\Pi}_{T+l}) \leq 1$. So we have $0 < \rho_{T+l} \leq \rho_{T+l}^y \leq 1$ for all $m \geq l \geq 1$.

Assume $m \geq l \geq 2$. Then by a) of this Lemma and (5.27), we have

$$\rho_{T+l} < 1 \iff \rho_{T+l}^y < 1 \quad \text{for all } m \geq l \geq 2.$$

Note that if $\rho_t \in (0,1)$ for all $t \leq T$, we have $G_i(y; \mathbf{\Pi}_T) > 0$ by d) of Lemma 5.2. Therefore

$$\rho_{T+1} < 1 \iff \rho_{T+1}^y < 1.$$

c) From (5.27) we obtain that for all $m \geq l \geq 1$

$$(1 - \rho_{T+l}) G_i(y, \mathbf{\Pi}_{T+l-1}) = G_i(y, \mathbf{\Pi}_{T+l}) - \rho_{T+l}.$$

This together with the basic recursion (4.13) for $\mathbf{\Pi}_t$ shows that Q' fulfills the recursion

$$\begin{aligned}
&Q'(a; y, i+1, \mathbf{\Pi}_{T+l}) = \frac{\mathbf{\Pi}_{T+l}(a; i+1)}{G_i(y, \mathbf{\Pi}_{T+l})} \\
&= \frac{(1 - \rho_{T+l}) \mathbf{\Pi}_{T+l-1}(a; i+1)}{G_i(y, \mathbf{\Pi}_{T+l})} + \frac{\rho_{T+l}\, \mathbf{W}_{T+l-1}(a; i+1)}{G_i(y, \mathbf{\Pi}_{T+l})} \\
&= \frac{(1 - \rho_{T+l}) G_i(y, \mathbf{\Pi}_{T+l-1})}{G_i(y, \mathbf{\Pi}_{T+l})} Q'(a; y, i+1, \mathbf{\Pi}_{T+l-1}) + \rho_{T+l}^y\, \mathbf{W}_{T+l-1}(a; i+1) \\
&= (1 - \rho_{T+l}^y) Q'_{\mathbf{\Pi}_{T+l-1}}(a; i+1, y) + \rho_{T+l}^y\, \mathbf{W}_{T+l-1}(a; i+1).
\end{aligned}$$

d) From part c) and Lemma 5.1 (5.15) with $q_0 = Q'(a; y, i+1, \mathbf{\Pi}_T)$ we obtain

$$Q'(a; y, i+1, \mathbf{\Pi}_{T+m}) = w - \big(w - Q'(a; y, i+1, \mathbf{\Pi}_T)\big) \prod_{l=1}^{m} (1 - \rho_{T+l}^y).$$

Since

$$w - \big(w - Q'(a; y, i+1, \mathbf{\Pi}_T)\big) \prod_{l=1}^{m} (1 - \rho_{T+l}^y)$$

is increasing as ρ_{T+l}^y, then by part b) the remaining holds. □

Note that $[\mathbf{W}_T(a; i+1) = \cdots = \mathbf{W}_{T+m-1}(a; i+1) = 1]$ or

$$[\forall l = 0, \ldots, m-1 \ \forall s \in \mathbf{X}_{T+l} \cup \mathbf{M}_{T+l} \quad s_{|i} = y]$$

fulfill the condition (5.23) in Lemma 5.6. Lemmas 5.5 and 5.6 take pivotal roles in the subsequent proofs. In the sequel, if we need the bounds of Q, we first use Lemma 5.6 to get a suitable bound of Q', and then bound Q through the bounding functions ℓ and $\hbar$ by Lemma 5.5. For example, we may need to calculate the lower bound of $Q(a; y, i+1, \mathbf{\Pi}_{T+m})$ in the case $\mathbf{W}_T(a; i+1) = \cdots = \mathbf{W}_{T+m-1}(a; i+1) = 1$. Then, by d) of Lemma 5.6, we have

$$\begin{aligned} Q'(a; y, i+1, \mathbf{\Pi}_{T+m}) &\geq 1 - \big(1 - Q'(a; y, i+1, \mathbf{\Pi}_T)\big) \prod_{l=1}^{m} (1 - \rho_{T+l}) \\ &\geq 1 - \prod_{l=1}^{m} (1 - \rho_{T+l}). \end{aligned}$$

By a) of Lemma 5.5 and a) of Lemma 5.4, we have

$$\begin{aligned} Q(a; y, i+1, \mathbf{\Pi}_{T+m}) &\geq \ell\big[Q'(a; y, i+1, \mathbf{\Pi}_{T+m})\big] = \ell\Big[1 - \prod_{l=1}^{m} (1 - \rho_{T+l})\Big] \\ &= 1 - \hbar\Big[\prod_{l=1}^{m} (1 - \rho_{T+l})\Big]. \end{aligned} \tag{5.28}$$

5.3 On the reachability of optimal solutions

In this Section, we investigate conditions which guarantee to find an optimal solution in finitely many iterations. We first review the related work in the literature with details. Then we present a unified Theorem which extends all these findings to our more general framework.

5.3.1 Related work

The first literature who considered conditions for finite reachability in the theoretical analysis of MBS are [Gut00] and [Gut03]. In this two papers, W. J. Gutjahr showed that

$$\mathbf{P}\big[\tau < \infty\big] \to 1 \quad \text{as} \quad \rho \to 0 \text{ or } N \to \infty$$

for his GBAS algorithm, see (5.1) for a definition of τ. GBAS can be described by our framework with a strategy of $\rho_t \equiv \rho > 0$, $\mathcal{M} = \text{GTMU}$ (global truncate memory), $S = \text{TS}$ (truncate selection), $\mathcal{L} = \text{WL}$ (weighted learning) and $M = N_b = 1$ (M the memory size and N_b the subsample size), see also Table 4.1. Another literature concerning conditions for finite reachability is [CJK07]. In which A. Costa et al inspected a CE algorithm on an unconstrained underlying CO instance. They showed that

$$\sum_{t=1}^{\infty} \prod_{m=1}^{t} (1-\rho_m)^L = \infty \;\Rightarrow\; \mathbf{P}\big[\tau < \infty\big] = 1$$

and if there is a unique optimal solution i.e. $|S^*| = 1$,

$$\mathbf{P}\big[\tau < \infty\big] = 1 \;\Rightarrow\; \sum_{t=1}^{\infty} \prod_{m=1}^{t} (1-\rho_m) = \infty.$$

In [WK14b], we considered a more general CE algorithm which covers the essential part of AS (ant system). We showed in [WK14b] that the conditions in [CJK07] for finite reachability still holds in that more general algorithm.

5.3.2 A unified theorem for reachability of optimal solutions

Theorem 5.7 below continues our former work in [WK14b]. It shows that the conditions for finite reachability presented in that publication still hold in the unified framework. Note that this Theorem does not require the details of the memory update rules, the subsample selection rules or the learning rules. Therefore, it applies to all algorithms covered by the framework.

Theorem 5.7. ***a)*** *If* $\sum_{t=1}^{\infty} \prod_{m=1}^{t} (1-\rho_m)^L = \infty$ *then* $\mathbf{P}(\tau < \infty) = 1$.

b) *If* $\mathbf{P}(\tau < \infty) = 1$ *then* $\sum_{t=1}^{\infty} \prod_{m=1}^{t} (1-\rho_m) = \infty$.

c) *If* $\rho_t \equiv \rho > 0$ *is a constant, then we have* $\mathbf{P}(\tau < \infty) < 1$, *but* $\mathbf{P}(\tau < \infty) \to 1$ *if either* $N \to \infty$ *or* $\rho \to 0$.

proof of Theorem 5.7. **a)** We fix an arbitrary optimal solution $s^* = \big(s_1^*, \dots, s_L^*\big) \in S^*$, then by definition (5.1) we have

$$\begin{aligned} \mathbf{P}(\tau = \infty) &= \mathbf{P}\Big(\bigcap_{t=0}^{\infty} [S^* \cap \mathbf{X}_t = \emptyset]\Big) \leq \mathbf{P}\Big(\bigcap_{t=0}^{\infty} [s^* \notin \mathbf{X}_t]\Big) \\ &= \mathbf{P}(s^* \notin \mathbf{X}_0) \prod_{t=1}^{\infty} \mathbf{P}\big[s^* \notin \mathbf{X}_t \mid s^* \notin \mathbf{X}_m, m = 0, \dots, t-1\big] \end{aligned} \tag{5.29}$$

We now derive an upper bound for the factors in (5.29). First we have

$$\begin{aligned} &\mathbf{P}\big[s^* \notin \mathbf{X}_t \mid s^* \notin \mathbf{X}_m, m = 0, \dots, t-1\big] \\ &\quad = \mathbf{E}\Big[\mathbf{P}\big[s^* \notin \mathbf{X}_t \mid \mathbf{\Pi}_t\big] \;\Big|\; s^* \notin \mathbf{X}_m, m = 0, \dots, t-1\Big]. \end{aligned} \tag{5.30}$$

Observe that for all $a \in A, y \in R_{i-1}$, $i \in \{1, \dots, L\}$ and $t \geq 0$, we have

$$Q(a; y, i, \mathbf{\Pi}_t) = \frac{\mathbf{\Pi}_t(a; i) C_{i-1}(y, a)}{\sum_{a' \in C_{i-1}(y)} \mathbf{\Pi}_t(a'; i) C_{i-1}(y, a')} \geq \mathbf{\Pi}_t(a; i) C_{i-1}(y, a)$$

because of the fact that $0 \leq \sum_{a' \in C_{i-1}(y)} \mathbf{\Pi}_t(a'; i) C_{i-1}(y, a') \leq 1$. And by (5.14) of Lemma 5.1, we know that

$$\mathbf{\Pi}_t(y; a) \geq \mathbf{\Pi}_0(a; i) \prod_{m=1}^{t} (1 - \rho_m).$$

As the solutions are sampled i.i.d. we may now conclude

$$\begin{aligned} &\mathbf{P}\big[s^* \notin \mathbf{X}_t \mid \mathbf{\Pi}_t\big] \\ &= \Big(\mathbf{P}\big[s^* \neq \mathbf{X}_t^{(1)} \mid \mathbf{\Pi}_t\big]\Big)^N = \Big(1 - \mathbf{P}\big[s^* = \mathbf{X}_t^{(1)} \mid \mathbf{\Pi}_t\big]\Big)^N \\ &= \Big(1 - Q(s_1^*; \diamond, 1, \mathbf{\Pi}_t) \prod_{i=2}^{L} Q\big(s_i^*; (s_1^*, \dots, s_{i-1}^*), i, \mathbf{\Pi}_t\big)\Big)^N \\ &\leq \left(1 - \mathbf{\Pi}_t(s_1^*; 1) C_0(\diamond, s_1^*) \prod_{i=2}^{L} \Big[\mathbf{\Pi}_t(s_i^*; i)\; C_{i-1}\Big((s_1^*, \dots, s_{i-1}^*), s_i^*\Big)\Big]\right)^N \\ &= \Big(1 - c(s^*) \prod_{i=1}^{L} \mathbf{\Pi}_t(s_i^*; i)\Big)^N \\ &\leq \left(1 - c(s^*) \prod_{i=1}^{L} \Big[\mathbf{\Pi}_0(s_i^*; i) \prod_{m=1}^{t} (1 - \rho_m)\Big]\right)^N \\ &\leq \Big(1 - \delta(s^*) \prod_{m=1}^{t} (1 - \rho_m)^L\Big)^N \end{aligned} \tag{5.31}$$

where

$$c(s^*) := \prod_{i=0}^{L-1} c_i\big((s_1^*, \dots, s_i^*); s_{i+1}^*\big) > 0$$

under assumption (5.4), and

$$\delta(s^*) := c(s^*) \prod_{i=1}^{L} \mathbf{\Pi}_0\big(s_i^*; i\big) > 0$$

under assumption (5.2).

For the first factor in (5.29) we have from (5.31) for $t = 0$

$$\mathbf{P}(s^* \notin \mathbf{X}_0) \leq \Big(1 - \delta(s^*)\Big)^N.$$

Combining these results we obtain

$$\begin{aligned} \mathbf{P}(\tau < \infty) &= 1 - \mathbf{P}(\tau = \infty) \\ &\geq 1 - \Big[\prod_{t=0}^{\infty} \Big(1 - \delta(s^*) \prod_{m=1}^{t} (1 - \rho_t)^L\Big)\Big]^N. \end{aligned} \tag{5.32}$$

When

$$\sum_{t=1}^{\infty} \prod_{m=1}^{t} (1 - \rho_m)^L = \infty,$$

by a) of Lemma 5.1 we have

$$\Big[\prod_{t=0}^{\infty} \Big(1 - \delta(s^*) \prod_{m=1}^{t} (1 - \rho_t)^L\Big)\Big]^N = 0.$$

Which in turn implies $\mathbf{P}[\tau < \infty] = 1$.

b) By assumption (5.5), we arbitrarily pick an $s \in S - S^*$ which is compatible to the feasibility constructions. We write $\mathbf{X}_t^{(\cdot)} \equiv s$ for $\mathbf{X}_t^{(n)} = s, n = 1, \ldots, N$ and $\mathfrak{S}$ for the event $\mathbf{X}_m^{(\cdot)} \equiv s$ for all $m = 0, \ldots, t-1$. Then, as samples are i.i.d.,

$$\begin{aligned} \mathbf{P}\big[\mathbf{X}_t^{(\cdot)} \equiv s, t = 0, 1, \ldots\big] &= \mathbf{P}\big[\mathbf{X}_0^{(\cdot)} \equiv s\big] \prod_{t=1}^{\infty} \mathbf{P}\big[\mathbf{X}_t^{(\cdot)} \equiv s \;\big|\; \mathbf{X}_m^{(\cdot)} \equiv s \text{ for } m = 0, \ldots, t-1\big] \\ &= \mathbf{P}\big[\mathbf{X}_0^{(1)} = s\big]^N \cdot \prod_{t=1}^{\infty} \mathbf{P}\big[\mathbf{X}_t^{(1)} = s \;\big|\; \mathfrak{S}\;\big]^N \end{aligned} \tag{5.33}$$

We have $\mathbf{P}(\mathbf{X}_0^{(1)} = s) = Q(s; \mathbf{\Pi}_0)$ for the first factor. Using the lower bound in Lemma

5.5 a) we obtain for the other factors in (5.33) with $s_{|i} := (s_1, \dots, s_i)$

$$\begin{aligned}
&\mathbf{P}\big[\mathbf{X}_t^{(1)} = s \;\big|\; \mathfrak{S}\,\big] = \mathbf{P}\big[\mathbf{X}_t^{(1)}(1) = s_1 \;\big|\; \mathfrak{S}\,\big] \\
&\quad \cdot \prod_{i=2}^{L} \mathbf{P}\big[\mathbf{X}_t^{(1)}(i) = s_i \;\big|\; \mathbf{X}_t^{(1)}(1,\dots,i-1) = s_{|i-1},\ \mathfrak{S}\,\big] \\
&= \mathbf{E}\big[Q(s_1; \diamond, 1, \mathbf{\Pi}_t) \;\big|\; \mathfrak{S}\,\big] \\
&\quad \cdot \prod_{i=2}^{L} \mathbf{E}\big[Q(s_i; s_{|i-1}, i, \mathbf{\Pi}_t) \;\big|\; \mathbf{X}_t^{(1)}(1,\dots,i-1) = s_{|i-1},\ \mathfrak{S}\,\big] \\
&\geq \prod_{i=1}^{L} \mathbf{E}\big[\ell\big(Q'(s_i; s_{|i-1}, i, \mathbf{\Pi}_t)\big) \big|\; \mathfrak{S}\,\big]
\end{aligned}$$

with $s_{|0} = \diamond$.

Since $\mathbf{M}_0 = \emptyset$ and $\mathbf{M}_{m+1}$ is a subsample of $\mathbf{M}_m \cup \mathbf{X}_m$ for each $m \in \mathbb{N}$, under the condition $\mathfrak{S}$ we have

$$\mathbf{M}_m \cup \mathbf{X}_m = (s, \dots, s) \text{ for all } m = 0, \dots, t-1.$$

This in turn implies that

$$\mathcal{N}_m^b = (s, \dots, s) \text{ for all } m = 0, \dots, t-1.$$

Therefore, with the assumption (5.9) $\mathbf{W}_m(s_i; i)$ all equal 1 for all $i = 1, \dots, L$ and all $m = 0, \dots, t-1$. Hence we may use Lemma 5.6 d) to obtain

$$Q'(s_i; s_{|i-1}, i, \mathbf{\Pi}_t) \geq 1 - \prod_{m=1}^{t} (1 - \rho_m)$$

for $t \geq 1$ and $i = 1, \dots, L$. Now, from (5.33) we obtain using Lemma 5.4 a)

$$\begin{aligned}
&\mathbf{P}\big[\mathbf{X}_t^{(\cdot)} \equiv s, t = 0, 1, \dots\big] \qquad (5.34) \\
&\geq Q(s, \mathbf{\Pi}_0)^N \prod_{t=1}^{\infty} \Big[\prod_{i=1}^{L} \ell\big(1 - \prod_{m=1}^{t} (1 - \rho_m)\big)\Big]^N \\
&= Q(s, \mathbf{\Pi}_0)^N \Big[\prod_{t=1}^{\infty} \Big(1 - h\big(\prod_{m=1}^{t} (1 - \rho_m)\big)\Big)\Big]^{LN}
\end{aligned}$$

By the assumption (5.2), we know that $Q(s, \mathbf{\Pi}_0) > 0$. And by (5.34) $\mathbf{P}(\tau < \infty) = 1$ requires

$$0 = Q(s, \mathbf{\Pi}_0)^N \Big[\prod_{t=1}^{\infty} \Big(1 - h\big(\prod_{m=1}^{t} (1 - \rho_m)\big)\Big)\Big]^{LN}.$$

As a result, $\mathbf{P}(\tau < \infty) = 1$ requires

$$0 = \Big[\prod_{t=1}^{\infty}\Big(1 - h(\prod_{m=1}^{t}(1-\rho_m))\Big)\Big]^{LN}$$

which in turn implies

$$\sum_{t=1}^{\infty}\hbar\Big(\prod_{m=1}^{t}(1-\rho_m)\Big) = \infty.$$

By Lemma 5.4 c) this is equivalent to

$$\sum_{t=1}^{\infty}\prod_{m=1}^{t}(1-\rho_m) = \infty$$

since

$$x_t := \prod_{m=1}^{t}(1-\rho_m) \to \prod_{m=1}^{\infty}(1-\rho_m) \quad \text{as } t \to \infty.$$

c) Obviously if $\rho_t \equiv \rho > 0$, $\mathbf{P}(\tau < \infty) < 1$ since $\sum_{t=1}^{\infty}(1-\rho)^t = \frac{1-\rho}{\rho} < \infty$. By (5.32), in this case

$$\mathbf{P}(\tau < \infty) = 1 - \mathbf{P}(\tau = \infty) \geq 1 - \Big[\prod_{t=0}^{\infty}\Big(1 - \delta(s^*)(1-\rho)^{t\cdot L}\Big)\Big]^N.$$

Note that when $\rho > 0$,

$$1 > \prod_{t=0}^{\infty}\Big(1 - \delta(s^*)(1-\rho)^{t\cdot L}\Big) \geq 0.$$

Hence $\mathbf{P}(\tau < \infty) \to 1$ as $N \to \infty$. Note also that

$$\prod_{t=0}^{\infty}\Big(1 - \delta(s^*)(1-\rho)^{t\cdot L}\Big) \to 0 \text{ as } \rho \to 0,$$

so as $\rho \to 0$, $\mathbf{P}(\tau < \infty) \to 1$. □

The conditions in a) and b) of Theorem 5.7 may require us to reduce the learning rates fast, e.g.

$$\rho_m = 1 - \sqrt[L]{\frac{m}{m+1}} \quad \text{for all } m \geq 1. \tag{5.35}$$

Recall that in each iteration, ρ_{t+1} reflects the relative importance of the learned empirical distribution $\mathbf{W}_t$ in the construction of the next model $\mathbf{\Pi}_{t+1}$, see the basic recursion (4.21). The empirical distribution $\mathbf{W}_t$ actually concentrates on a local area closely around $\mathcal{N}_t^b$. Therefore, ρ_{t+1} can be seen as a measure for the strength of a local exploitation in that area in iteration $t+1$. A higher ρ_{t+1} makes algorithm emphasize more on the local exploitation. Reducing the learning rates fast means that the search emphasis may gradually shift from local exploitation to a biased global exploration as time goes. This makes the algorithm escape from a local trap more easily in the later stage, hence guarantees to reach an optimal solution in finitely many iterations.

5.4 Some runtime analysis results

The analysis in Section 5.3 assumes an arbitrary CO instance. Therefore, the results are universal i.e. do not depend on problems. A drawback in this kind of analysis is that we can not give any insight to the bound of τ even in the case that finite reachability is guaranteed.

To give some insights to the bound of τ, we need to do a runtime analysis. This Section gives two runtime results for the framework. Due to the popularity of the case $\rho_t \equiv \rho > 0$ in practice, we shall assume a constant learning rate $\rho_t \equiv \rho > 0$ in this Section.

5.4.1 Definitions of two test problems

As mentioned, in runtime analysis, we need assume a particular test problem. The frequently used test problems are LeadingOne and OneMax, see e.g. [NW06], [DNSW07], [DJ07], [NW09], [Gut07], [Gut08], [CTCY10] and [WK14b]. We now formally define these two problems.

In OneMax problem and LeadingOne problem, we consider the unconstrained case with alphabet $\{0,1\}$, i.e. a CO instance $(S, f, 0)$ with $S = \{0,1\}^L$. And we use the non-greedy feasibility construction, i.e. $Q\big(a; y, i, \mathbf{\Pi}_t\big) = \mathbf{\Pi}_t(a; i)$ for all $a \in \{0,1\}$, $y \in \{0,1\}^{i-1}$ and $i = 1, \ldots, L$. The cost function for OneMax is defined as

$$f(s) := L - \sum_{i=1}^{L} s_i \quad \text{for all } s = (s_1, \ldots, s_L) \in S. \tag{5.36}$$

Note that minimizing (5.36) is equivalent to maximizing the number of $1s$ in a solution. Obviously, the unique optimal solution is $(1, 1, \ldots, 1)$ which has a cost value 1.

For LeadingOne problem, the corresponding cost function is defined as

$$f(s) := L - \sum_{l=1}^{L} \prod_{i=1}^{l} s_i + 1 \quad \text{for all } s = (s_1, \ldots, s_L) \in S. \tag{5.37}$$

Minimizing (5.37) is intrinsically equivalent to maximizing the number of consecutive 1s counted from the left of the solution. And the unique optimal solution is again $(1, 1, \ldots, 1)$ with a cost value 1.

5.4.2 Runtime results for unrestricted models

From Theorem 5.7 c), we know that for the case of $\rho_t \equiv \rho > 0$, finite reachability can not be guaranteed i.e. $\mathbf{P}\big[\tau = \infty\big] > 0$. This results in an *infinite* expected runtime. However, it does not mean that this setting is harmful. In [CTCY10] for a UMDA algorithm (i.e. our framework with $\rho_t \equiv \rho = 1$, non-memory, truncate selection for subsample, and uniform learning) on LeadingOne problem, Chen et al showed that if we

take sample size $N = L^{2+\epsilon}$ for some constant $\epsilon \in (0,1)$ and subsample size $N_b = \beta \cdot N$ for a constant $\beta \in (0,1)$, then

$$\mathbf{P}\big[\tau < 4 \cdot L\big] \to 1 \quad \text{as } L \to \infty.$$

This means that with a large probability, the runtime of the UDMA on LeadingOne is $O(L^{3+\epsilon})$.

The following Theorem is a runtime result proposed in our former work [WK14b]. It extends the results in [CTCY10] to a more general case that $\rho_t \equiv \rho > 0$. The Theorem essentially states that if we take $\rho_t \equiv \rho \in (0,1)$ and let the sample size grow as $N = L^{2+\epsilon}$ for some $\epsilon > 0$, then we may reach an optimal solution in L iterations with a probability converging to 1, i.e. the runtime is $O(L^{3+\epsilon})$ in a stochastic sense.

Theorem 5.8. *Suppose that we do not use memory ($\mathcal{M} = NM$), select the subsample $\mathcal{N}_t^b$ with truncate selection ($\mathcal{S} = TS$), employ the uniform learning ($\mathcal{L} = UL$) and unconstrained non-greedy feasibility distributions. Let $\rho_t \equiv \rho \in (0,1)$, sample size $N = L^{2+\epsilon}$ for some $\epsilon > 0$ and $N_b = \lfloor \beta N \rfloor$ for some $0 < \beta < \frac{1}{3e} \prod_{m=1}^{\infty} \big(1 - (1-\rho)^m\big)$. Let each $\mathbf{\Pi}_0(1,i) = \frac{1}{2}$, i.e. we start with the uniform distribution. Then for a LeadingOne problem, defined in* (5.37)*, we have $\mathbf{P}(\tau < L) \to 1$ as $L \to \infty$.*

Here, recall that with $\mathcal{M} = \text{NM}$, we identify $\mathbf{M}_t \equiv \emptyset$ for all $t \in \mathbb{N}$; with $\mathcal{S} = \text{TS}$, we select N_b best solutions in $\mathbf{M}_t \cup \mathbf{X}_t$ as the subsample $\mathcal{N}_t^b$; with $\mathcal{L} = \text{UL}$, we use the relative frequencies as $\mathbf{W}_t$, i.e.

$$\mathbf{W}_t(a;i) = \frac{\sum_{s=(b_1,\ldots,b_L) \in \mathcal{N}_t^b} \mathbb{1}_{\{a\}}(b_i)}{N_b}$$

for each $a \in \mathcal{A} = \{0,1\}$ and $i = 1,\ldots,L$; in unconstrained non-greedy feasibility construction,

$$C_i(y;a) = \frac{1}{|\mathcal{A}|} = \frac{1}{2}$$

for each feasible partial string y and letter $a \in \mathcal{A}$, and this makes $Q(a;y,i+1,\mathbf{\Pi}_t) = \mathbf{\Pi}_t(a;i+1)$ for each $t = 0,1,2,\ldots$.

Proof of Theorem 5.8. See also [WK14b]. With $\mathcal{A} = \{0,1\}$, we write $\pi_t(i) := \mathbf{\Pi}_t(1;i)$ for all $t \in \mathbb{N}$ and $i = 1,\ldots,L$. As we consider the unconstrained case and use non-greedy feasibility distributions here, $Q(1;y,i,\mathbf{\Pi}_t) = \pi_t(i)$ for all $i = 0,\ldots,L-1$, $y \in \{0,1\}^i$ and $t \in \mathbb{N}$. Then the sample $\mathbf{X}_t = (\mathbf{X}_t^{(1)},\ldots,\mathbf{X}_t^{(N)})$ can be viewed as a random matrix with mutually independent entries and solution $\mathbf{X}_t^{(n)}$ as n-th row. We denote by $\mathbf{X}_t^{[\cdot]} = \big(\mathbf{X}_t^{[1]},\ldots,\mathbf{X}_t^{[N]}\big)$ the corresponding ordered matrix with rows ordered according to increasing cost function values :

$$f(\mathbf{X}_t^{[1]}) \leq \cdots \leq f(\mathbf{X}_t^{[N]}).$$

Since we use NM memory update and TS subsample selection, $\mathcal{N}_t^b$ is the sub-matrix of the first N_b rows of $\mathbf{X}_t^{[\cdot]}$. Because we use UL learning, the empirical distributions from $\mathcal{N}_t^b$ may be then written as

$$\mathbf{W}_t(1;i) = \frac{1}{N_b}\sum_{n=1}^{N_b} \mathbf{X}_t^{[n]}(i) =: w_t(i)$$

and $w_t := (w_t(1), \ldots, w_t(L))$.

For the proof we let $w_t, t = 0, 1, \ldots$ move over a sequence of increasing levels w_t^*. These levels are defined in such a way that the LeadingOne cost value $f(\mathbf{X}_t)$ is strictly decreasing and at the same time the update of π_t can be controlled such that we are able to give a lower bound on the probability for this to happen.

For $t = 0, \ldots, L-1$ let

$$w_t^* := \big(w_t^*(1), \ldots, w_t^*(L)\big) := (1, \ldots, 1, \alpha_t, \ldots, \alpha_t) \tag{5.38}$$

with $t+1$ entries '1' and some $\alpha_t \in (0,1)$ to be defined below. We write $w_t \succeq w_t^*$ if and only if $w_t(i) = 1$ for $i = 1, \ldots, t+1$ and $w_t(i) \geq \alpha_t$ for $i = t+2, \ldots, L$. For $t = L-1$, $w_t \succeq w_t^*$ means that $\mathcal{N}_t^b$ consists of the optimal solution $s^* = (1, \ldots, 1)$ only. Hence we have

$$\begin{aligned}\mathbf{P}(\tau < L) &\geq \mathbf{P}(w_{L-1} \succeq w_{L-1}^*) \geq \mathbf{P}\big(w_0 \succeq w_0^*, \ldots, w_{L-1} \succeq w_{L-1}^*\big) \\ &= \mathbf{P}(w_0 \succeq w_0^*) \prod_{t=1}^{L-1} \mathbf{P}\big[w_t \succeq w_t^* \mid w_m \succeq w_m^*, m = 0, \ldots, t-1\big].\end{aligned} \tag{5.39}$$

We show below that there are constants $a, b, c > 0$ such that for all L large enough and for all N

$$\begin{aligned}&\mathbf{P}\big[w_t \succeq w_t^* \mid w_m \succeq w_m^*, m = 0, \ldots, t-1\big] \\ &\quad \geq (1 - e^{-aN})\big(1 - e^{-b\frac{N}{L^2}+c}\big)^{L-t-1}.\end{aligned} \tag{5.40}$$

With $N = L^{2+\epsilon}$ for some $\epsilon > 0$ we may then conclude that

$$\mathbf{P}(\tau < L) \geq (1 - e^{-aL^{2+\epsilon}})^L \big(1 - e^{-bL^{\epsilon}+c}\big)^{(L^2-L)/2}.$$

Observe that

$$(1 - e^{-aL^{2+\epsilon}})^L = e^{L\,\ln(1-e^{-aL^{2+\epsilon}})} = e^{-\frac{L}{e^{aL^{2+\epsilon}}}\,\frac{\ln(1-e^{-aL^{2+\epsilon}})}{-e^{-aL^{2+\epsilon}}}} \to 1 \text{ as } L \to \infty,$$

and similarly

$$\big(1 - e^{-bL^{\epsilon}+c}\big)^{(L^2-L)/2} \to 1 \text{ as } L \to \infty.$$

Hence the proof of Theorem 5.8 is complete once we have shown (5.40) for $t = 0, \ldots, L-1$.

In the first step we show how the levels w_t^* influence π_t. We abbreviate the event $[w_m \succeq w_m^*, m = 0, \ldots, t-1]$ by $\mathfrak{W}_t$, $\mathfrak{W}_0$ indicating the empty condition. Let $\alpha_t := \frac{1}{2}(1-\frac{1}{L})^{t+1}, t = 0, \ldots, L-1$ and $\alpha_{-1} := \frac{1}{2}$. Conditioned on $\mathfrak{W}_t$,

$$\pi_t(i) \geq \begin{cases} 1-(1-\rho)^{t-i+1} & \text{for } 1 \leq i \leq t \\ \alpha_{t-1} & \text{for } t < i \leq L \end{cases} \tag{5.41}$$

for $t = 0, \ldots, L-1$. The proof is by induction on t using the basic recursion (4.21) and the property

$$\begin{aligned} w_m(i) &= 1 \quad \text{for all } m = i-1, i, \ldots, t-1 \text{ if } i \leq t, \\ w_m(i) &\geq \alpha_m \quad \text{for all } m = 0, 1, \ldots, t-1 \text{ if } i > t, \end{aligned}$$

which follows from the definition of w_t^* under $\mathfrak{W}_t$, see Table 5.1 below. Let $v_{t+1} =$

i	1	2	$\cdots$	i	$\cdots$	t	$t+1$	$\cdots$		L
π_0	$\frac{1}{2}$	$\frac{1}{2}$	$\cdots$	$\frac{1}{2}$	$\cdots$	$\frac{1}{2}$	$\frac{1}{2}$	$\cdots$		$\frac{1}{2}$
w_0^*	1	α_0	α_0	$\cdots$			α_0	$\cdots$		α_0
w_1^*	1	1	α_1	$\cdots$			α_1	$\cdots$		α_1
$\vdots$	$\vdots$		$\ddots$							$\vdots$
w_{i-1}^*	1	1	$\cdots$	1	α_{i-1}	$\cdots$	α_{i-1}	$\cdots$		α_{i-1}
$\vdots$	$\vdots$	$\vdots$		$\vdots$	$\ddots$			$\cdots$		$\vdots$
w_{t-1}^*	1	1	$\cdots$	1	$\cdots$	1	α_{t-1}	$\cdots$		α_{t-1}
w_t^*	1	1	$\cdots$	1	$\cdots$	1	1	α_t	$\cdots$	α_t

Table 5.1: The levels w_t^* for the empirical distributions

$\mathbf{P}\big[\mathbf{X}_t^{(n)}(1) = \cdots = \mathbf{X}_t^{(n)}(t+1) = 1 \mid \mathfrak{W}_t\big]$ be the probability to sample a solution that has $t+1$ leading 1s, then from (5.41) we obtain

$$\begin{aligned} v_{t+1} &= \prod_{i=1}^{t+1} \pi_t(i) \geq \alpha_{t-1} \prod_{i=1}^{t} \big(1-(1-\rho)^{t-i+1}\big) \\ &\geq \alpha_{L-1} \prod_{i=1}^{\infty} \big(1-(1-\rho)^{t-i+1}\big) \\ &\geq \frac{1}{3e} \prod_{m=1}^{\infty} \big(1-(1-\rho)^{m}\big) =: \kappa(\rho) \end{aligned} \tag{5.42}$$

for L large enough as then $\alpha_t \geq 1/(3e)$.

Now we want to determine simple conditions on $\mathbf{X}_t$ that imply $w(\mathbf{X}_t) \succeq w_t^*$. To do so, we look at the matrix $\mathbf{X}_t$ *columnwise* observing the independence of its entries.

Let $M^{(t+1)}$ be the number of rows with at least $t+1$ leading 1s, then $M^{(t+1)} \geq N_b$ implies $w_t(i) = 1, i = 1, \ldots, t+1$. These $M^{(t+1)}$ rows must be the first rows in $\mathbf{X}_t^{[\cdot]}$.

Next, the number of 1s in column $i = t + 2$ on these rows

$$Y^{(i)} := \sum_{n=1}^{M^{(i-1)}} \mathbf{X}_t^{[n]}(i) \;\geq\; \alpha_t N_b \tag{5.43}$$

implies $w_t(i) \geq \alpha_t$. Here we have to restrict the number of rows to the present 'candidate' rows $1, \ldots, M^{(i-1)}$ from which the set $\mathcal{N}_t^b$ is selected. After looking at column i in this way, we define

$$M^{(i)} := \max\{N_b, Y^{(i)}\} \tag{5.44}$$

as the updated number of present candiates for $\mathcal{N}_t^b$, and repeat (5.43), (5.44) for $i = t + 3, \ldots, L$. We then obtain

$$\begin{aligned} &\mathbf{P}\big[w_t \succeq w_t^* \mid \mathfrak{W}_t\big] \\ &\geq \mathbf{P}\big[M^{(t+1)} \geq N_b, Y^{(i)} \geq \alpha_t N_b, i = t+2, \ldots, L \mid \mathfrak{W}_t\big] \\ &= \mathbf{P}\big[M^{(t+1)} \geq N_b \mid \mathfrak{W}_t\big] \cdot \prod_{i=t+2}^{L} \mathbf{P}\big[Y^{(i)} \geq \alpha_t N_b \\ &\qquad \mid Y^{(l)} \geq \alpha_t N_b, l = t+2, \ldots, i-1, M^{(t+1)} \geq N_b, \mathfrak{W}_t\big] \end{aligned} \tag{5.45}$$

To derive the desired lower bounds for these expressions we need the Chernoff bound (see e.g. [RP95] Theorem 4.2 p. 70) in the following form: let $Z_1, \ldots, Z_m$ be i.i.d. 0-1-distributed with success probability p then for any $0 < r < mp$ we have

$$\mathbf{P}\big(\sum_{i=1}^{m} Z_i \leq r\big) \leq e^{-\frac{1}{2}(1-\frac{r}{mp})^2 mp}. \tag{5.46}$$

Conditioned on $\mathfrak{W}_t$, $M^{(t+1)}$ is distributed as the number of successes in a row of N i.i.d. experiments, each with success probability $p := v_{t+1}$. We obtain for $\beta < \kappa(\rho) \leq v_{t+1}$ that $N_b = \lfloor \beta N \rfloor \leq \beta N < v_{t+1} N$. Hence for $t = 0, \ldots, L-1$

$$\begin{aligned} \mathbf{P}\big[M^{(t+1)} \geq N_b \mid \mathfrak{W}_t\big] &= 1 - \mathbf{P}\big[M^{(t+1)} < N_b \mid \mathfrak{W}_t\big] \\ &\geq 1 - \mathbf{P}\big[M^{(t+1)} < \frac{\beta}{v_{t+1}} v_{t+1} N \mid \mathfrak{W}_t\big] \\ &\geq 1 - e^{-\frac{1}{2}(1-\frac{\beta}{\kappa(\rho)})^2 \kappa(\rho) N}, \end{aligned} \tag{5.47}$$

where we used (5.46). Hence, in (5.40) we may define $a := \frac{1}{2}(1 - \frac{\beta}{\kappa(\rho)})^2 \kappa(\rho)$. Similarly, $Y^{(t+2)}$ is distributed as the number of 1s in $M^{(t+1)}$ i.i.d. trials each with success probability $\pi_t(t+2)$. From (5.41) and the definition of α_t we see that under the condition used in (5.45) we have $\pi_t(t+2) M^{(t+1)} \geq \alpha_{t-1} M^{(t+1)} > \alpha_t N_b$. Using the Chernoff bound we therefore obtain for L large enough

$$\begin{aligned} \mathbf{P}\big[Y^{(t+2)} \geq \alpha_t N_b \mid \mathfrak{W}_t, M^{(t+1)} \geq N_b\big] &= 1 - \mathbf{P}\big[Y^{(t+2)} < \alpha_t N_b \mid \mathfrak{W}_t, M^{(t+1)} \geq N_b\big] \\ &\geq 1 - \mathbf{E}\big[e^{-\frac{1}{2}(1-\frac{\alpha_t N_b}{\pi_t(t+2) M^{(t+1)}})^2 \pi_t(t+2) M^{(t+1)}} \mid \mathfrak{W}_t, M^{(t+1)} \geq N_b\big] \\ &\geq 1 - e^{-\frac{1}{2}(1-\frac{\alpha_t}{\alpha_{t-1}})^2 \frac{N_b}{3e}} = 1 - e^{-\frac{N_b}{6eL^2}} \geq 1 - e^{-\frac{\beta N - 1}{6eL^2}} \end{aligned} \tag{5.48}$$

where we used $\frac{\alpha_t}{\alpha_{t-1}} = 1 - \frac{1}{L}$. A completely analogous derivation holds for the other factors in (5.45) with $i = t+3, \ldots, L$. Hence, with $b := \frac{\beta}{6e}, c := \frac{1}{6e}$, we see from (5.48), (5.47) and (5.45) that (5.40) holds for L large enough. □

Theorem 5.7 c) and (5.34) show that a constant learning rate reduces the global 'exploration' of the search space S. However, Theorem 5.8 shows that we can compensate for that by increasing the local 'exploitation' in terms of a growing sample size, at least in specific problems. A different example of such a balance shall be given in Theorem 6.8 in Chapter 6.

5.4.3 Runtime results for restricted models

The studies in [CTCY10] and [WK14b] complement those in [NW06], [NW09], [DNSW07], [DJ07] and [Gut08] who considered a theoretical MBS algorithm with restricted models on OneMax and LeadingOne. The MBS algorithm they considered is 1-ANT. Generally, 1-ANT samples only one solution in each iteration, and update the models similarly as $\mathcal{MMAS}$, i.e. for all $a \in \mathcal{A} = \{0,1\}$ and $i = 1, \ldots, L$,

$$\mathbf{\Pi}_{t+1}(a;i) := \begin{cases} \mathbf{\Pi}_t(a;i) & \text{if } f(s_t) > f(\mathbf{X}_{t-1}^{BF}), \\ \min\{\mathbf{\Pi}_t(a;i)(1-\rho) + \rho,\ p_{max}\} & \text{if } f(s_t) \le f(\mathbf{X}_{t-1}^{BF}) \text{ and } s_t(i) = 1, \\ \max\{\mathbf{\Pi}_t(a;i)(1-\rho),\ p_{min}\} & \text{if } f(s_t) \le f(\mathbf{X}_{t-1}^{BF}) \text{ and } s_t(i) = 0, \end{cases}$$

where $s_t = \big(s_t(1), \ldots, s_t(L)\big) \in S = \{0,1\}^L$ is the random solution sampled in iteration t, $\mathbf{X}_{t-1}^{BF}$ is the best solution found within iterations $0, 2, \ldots, t-1$, $\rho \in (0,1)$ is a fixed constant learning rate, $p_{min}, p_{max} \in (0,1)$ are fixed bounds for restricting the models, and f is the cost function defined as in (5.36) or (5.37).

In the literature, it is common to set $p_{min} = \frac{1}{L}$ and $p_{max} = 1 - \frac{1}{L}$. In [NW06] and [NW09], F. Neumann et al showed that when $\rho \ge \frac{1}{L^{1-\epsilon}}$ for some $\epsilon \in (0,1)$, the runtime of 1-ANT on OneMax problem is $O(L^2)$ with a probability approaching to 1 rapidly as $L \to \infty$; and when $\rho \le \frac{1}{L^{1+\epsilon}}$ for some $\epsilon > 0$, the runtime of 1-ANT on OneMax is exponentially also with a probability approaching to 1. [DNSW07] and [DJ07] continued the studies in [NW06] and [NW09] for the case that ρ is in the critical window $[\frac{1}{L^{1+\epsilon}}, \frac{1}{L^{1-\epsilon}}]$. Particularly, for the case $\rho \le 1/(L \log L)$, [DNSW07] and [DJ07] showed that 1-ANT are rather inefficient on both LeadingOne and OneMax i.e. the runtime can not be bounded above by any polynomial. In [Gut08], W. J. Gutjahr showed that for the case $\rho \ge 1 - \frac{1}{L}$, the expected runtime of 1-ANT on OneMax is $O(L \log L)$. Summarily, those literatures show that for a big constant learning rate $\rho \in (0,1)$, we may find an optimal solution efficiently if we reward models only with best solution found so far and employ lower and upper bounds to restrict models.

Theorem 5.9 below continues the studies in [NW06], [NW09], [DNSW07], [DJ07] and [Gut08]. It essentially states that in the case of $\rho_t \equiv \rho > 1 - \frac{1}{L}$ and $\mathcal{M}$= NM (i.e. $\mathbf{M}_t \equiv \emptyset$, reward models with present solutions), if we also employ restricted models, then, with an overwhelming probability, the runtime of the framework on OneMax is $O(L^{2+\epsilon})$ for some $\epsilon \in (0,1)$. Therefore, update of models only with best solution found so far is *not* so crucial in efficiently finding an optimal solution.

Theorem 5.9. *Let $\mathcal{M}$ = NM (non-memory), $\mathcal{S}$ = TS (truncate selection), $\mathcal{L}$ = UL (uniform learning), $\mathcal{A} = \{0,1\}$ and $\mathbf{\Pi}_0$ uniform on $S = \mathcal{A}^L$. We use unconstrained non-greedy feasibility construction, and employ a lower bound p_{min} in the update of models i.e. for all $t \geq \mathbb{N}$ after applying the basic recursion (4.21) , we reset $\mathbf{\Pi}_{t+1}(a;i) = p_{min}$ and $\mathbf{\Pi}_{t+1}(1-a;i) = 1-p_{min}$ for all $a \in \mathcal{A}$ and $i = 1,\ldots,L$ with $\mathbf{\Pi}_{t+1}(a;i) < p_{min} := \frac{1}{L}$. Then if $\rho_t \equiv \rho \geq 1 - \frac{1}{L}$, $N = L^{1+\epsilon}$ for an arbitrarily fixed $\epsilon > 0$ and $N_b = 1$, the runtime of the framework on OneMax is $O(L^{2+\epsilon})$ with a probability bigger than $1 - o(\beta^{L^\epsilon/2})$ for some constant $\beta \in (0,1)$.*

Proof. Let $\mathbf{X}_t^{IT} = \big(\mathbf{X}_t^{IT}(1),\ldots,\mathbf{X}_t^{IT}(L)\big) \in S = \{0,1\}^L$ be the best solution in iteration t. Note that we do not use memory here and $\mathcal{S}$ = TS with $N_b = 1$, the empirical distribution $\mathbf{W}_t$ would be learned from $\mathbf{X}_t^{IT}$ only. Note also that we consider unconstrained non-greedy feasibility construction here i.e.

$$Q(a;y,i+1,\mathbf{\Pi}_t) = \mathbf{\Pi}_t(a;i+1) \quad \text{for all } a \in \{0,1\}, i = 0,1,\ldots,L-1, y \in R_{i-1} \text{ and } t \in \mathbb{N}.$$

We first show two facts which would be rather involved in the proof of the Theorem.

Claim 5.10. *Under the assumption of Theorem 5.9, we have*

$$\mathbf{\Pi}_{t+1}(a;i) = \begin{cases} 1 - \frac{1}{L} & \text{if } a = \mathbf{X}_t^{IT}(i), \\ \frac{1}{L} & \text{if } a = 1 - \mathbf{X}_t^{IT}(i), \end{cases} \quad \text{for all } t \in \mathbb{N}, a \in \{0,1\}, i = 1,\ldots,L.$$

Proof of Claim 5.10. Now arbitrarily fix $t \in \mathbb{N}, i = 1,\ldots,L$. Since $\mathcal{L}$ = UL,

$$\mathbf{W}_t\big(\mathbf{X}_t^{IT}(i);i\big) = 1 \quad \text{and} \quad \mathbf{W}_t\big(1 - \mathbf{X}_t^{IT}(i);i\big) = 0.$$

Then by the basic recursion (4.21) and $\rho \geq 1 - \frac{1}{L}$, we have, for $a = 1 - \mathbf{X}_t^{IT}(i)$

$$\begin{aligned} \mathbf{\Pi}_{t+1}\big(a;i\big) &= (1-\rho)\mathbf{\Pi}_t(;i) + \rho\mathbf{W}_t(a;i) \\ &= (1-\rho)\mathbf{\Pi}_t(a;i) \leq (1-\rho) \leq \tfrac{1}{L} = p_{min}. \end{aligned}$$

Since we also employ the lower bound $p_{min} = \frac{1}{L}$ to adjust the model after applying basic recursion (4.21), we can conclude that

$$\mathbf{\Pi}_{t+1}(a;i) = \begin{cases} 1 - \frac{1}{L} & \text{if } a = \mathbf{X}_t^{IT}(i), \\ \frac{1}{L} & \text{if } a = 1 - \mathbf{X}_t^{IT}(i). \end{cases}$$

□

Claim 5.11. *Under the assumption of Theorem 5.9, we have that for any $l > 0$*

$$\mathbf{P}\big[f(\mathbf{X}_{t+1}^{IT}) \leq l-1 \;\big|\; f(\mathbf{X}_t^{IT}) = l\big] \geq 1 - \Big[1 - \frac{1}{L}\big(1 - \frac{1}{L}\big)^{L-1}\Big]^{L^{1+\epsilon}} > 0,$$

where f is the cost function of OneMax, see (5.36).

Proof of Claim 5.11. Let $\delta := \{i \mid \mathbf{X}_t^{IT}(i) = 1,\ i = 1, \ldots, L\} \subseteq \{1, 2, 3, \ldots, L\}$. Then by Claim 5.10, we have

$$\mathbf{\Pi}_{t+1}(a; i) = \begin{cases} 1 - \frac{1}{L} & \text{if } i \in \delta,\ a = 1, \\ \frac{1}{L} & \text{if } i \in \delta,\ a = 0, \\ 1 - \frac{1}{L} & \text{if } i \notin \delta,\ a = 0, \\ \frac{1}{L} & \text{if } i \notin \delta,\ a = 1. \end{cases} \tag{5.49}$$

Let $l > 0$ be arbitrarily fixed. Conditioned on $[f(\mathbf{X}_t^{IT}) = l]$, we have $\delta \neq \{1, 2, \ldots, L\}$ and $|\delta| = L - l$.

Observe that $[f(\mathbf{X}_{t+1}^{IT}) \leq l - 1]$ is implied by event

$$[\ \exists n = 1, \ldots, N :\ \forall i \in \delta\ \ \mathbf{X}_{t+1}^{(n)}(i) = 1 \quad \text{and} \quad \exists_1\ j \in \{1, \ldots, L\} - \delta\ \ \mathbf{X}_{t+1}^{(n)}(j) = 1\ ],$$

where notation $\exists_1$ means "exists exact one", and recall that $\mathbf{X}_t^{(n)}$ is the n-th random solution sampled from $\mathbf{\Pi}_{t+1}$ and $\mathbf{X}_t^{(n)}(i)$ indicates the i-th position of $\mathbf{X}_t^{(n)}$. We are to show a lower bound for the probability of this event conditioned on $[f(\mathbf{X}_t^{IT}) = l]$.

Note that for any fixed $n \in \{1, \ldots, N\}$, by (5.49) we have

$$\begin{aligned} &\mathbf{P}\big[\forall i \in \delta\ \mathbf{X}_{t+1}^{(n)}(i) = 1 \text{ and } \exists_1\ j \in \{1, \ldots, L\} - \delta\ \mathbf{X}_{t+1}^{(n)}(j) = 1 \ \big|\ f(\mathbf{X}_t^{IT}) = l\big] \\ &\quad = \mathbf{E}\Big[\big(1 - \frac{1}{L}\big)^{L-l}\ \cdot\ \binom{l}{1}\ \frac{1}{L}\big(1 - \frac{1}{L}\big)^{l-1}\ \Big|\ f(\mathbf{X}_t^{IT}) = l\Big] \\ &\quad \geq \frac{1}{L}\big(1 - \frac{1}{L}\big)^{L-1}, \end{aligned} \tag{5.50}$$

where, observe the fact that positions are independent of each other.

Note that $\mathbf{X}_{t+1}^{(1)}, \ldots, \mathbf{X}_{t+1}^{(N)}$ are *i.i.d.* By (5.50), we have

$$\begin{aligned} &\mathbf{P}\big[\exists n = 1, \ldots, N :\ \forall i \in \delta\ \mathbf{X}_{t+1}^{(n)}(i) = 1 \text{ and } \exists_1\ j \in \{1, \ldots, L\} - \delta\ \mathbf{X}_{t+1}^{(n)}(j) = 1\ \big|f(\mathbf{X}_t^{IT}) = l\big] \\ &\quad = 1 - \Big[1 - \mathbf{P}\big[\forall i \in \delta\ \mathbf{X}_{t+1}^{(1)}(i) = 1 \text{ and } \exists_1\ j \in \{1, \ldots, L\} - \delta\ \mathbf{X}_{t+1}^{(1)}(j) = 1\ \big|f(\mathbf{X}_t^{IT}) = l\big]\Big]^N \\ &\quad \geq 1 - \Big[1 - \frac{1}{L}\big(1 - \frac{1}{L}\big)^{L-1}\Big]^N = 1 - \Big[1 - \frac{1}{L}\big(1 - \frac{1}{L}\big)^{L-1}\Big]^{L^{1+\epsilon}} > 0. \end{aligned}$$

As a result, we have

$$\mathbf{P}\big[f(\mathbf{X}_{t+1}^{IT}) \leq l - 1 \ \big|\ f(\mathbf{X}_t^{IT}) = l\big] \geq 1 - \Big[1 - \frac{1}{L}\big(1 - \frac{1}{L}\big)^{L-1}\Big]^{L^{1+\epsilon}} > 0.$$

□

Now, we return to the proof of Theorem 5.9. By Claim 5.11, we first show a lower bound of the probability for event $[\tau \leq L]$, see (5.1) for a definition of τ. Obviously,

$$[\ \forall L - 1 \geq t \geq 0 \quad f(\mathbf{X}_t^{IT}) \leq L - t - 1\ ]$$

implies $[\tau \le L]$. Note that

$$\mathbf{P}\big[\forall L-1 \ge t \le 0 \quad f(\mathbf{X}_t^{IT}) \le L-t-1\big] \tag{5.51}$$

$$= \mathbf{P}\big[f(\mathbf{X}_0^{IT}) \le L-1\big] \prod_{t=1}^{L-1} \mathbf{P}\Big[f(\mathbf{X}_t^{IT}) \le L-t-1 \;\Big|\; \forall t-1 \ge m \ge 0 \tag{5.52}$$

$$f(\mathbf{X}_m^{IT}) \le L-m-1\Big]$$

$$= \mathbf{P}\big[f(\mathbf{X}_0^{IT}) \le L-1\big] \prod_{t=1}^{L-1} \mathbf{P}\Big[f(\mathbf{X}_t^{IT}) \le L-t-1 \;\Big|\; f(\mathbf{X}_{t-1}^{IT}) \le L-t\Big].$$

Note that the first factor in (5.51) is

$$\mathbf{P}\big[f(\mathbf{X}_0^{IT}) \le L-1\big] = 1 - 2^{-L\cdot N} = 1 - 2^{-L^{2+\epsilon}}, \tag{5.53}$$

because $\mathbf{\Pi}_0$ is uniform on $\{0,1\}^L$. Each factor in the finite products in (5.51) can be further written as

$$\mathbf{P}\Big[f(\mathbf{X}_t^{IT}) \le L-t-1 \;\Big|\; f(\mathbf{X}_{t-1}^{IT}) \le L-t\Big] \tag{5.54}$$

$$= \mathbf{P}\Big[f(\mathbf{X}_t^{IT}) \le L-t-1 \;\Big|\; 0 < f(\mathbf{X}_{t-1}^{IT}) \le L-t\Big]$$

$$\cdot \mathbf{P}\Big[0 < f(\mathbf{X}_{t-1}^{IT}) \le L-t \;\Big|\; f(\mathbf{X}_{t-1}^{IT}) \le L-t\Big]$$

$$+ \mathbf{P}\Big[f(\mathbf{X}_t^{IT}) \le L-t-1 \;\Big|\; f(\mathbf{X}_{t-1}^{IT}) = 0\Big]\mathbf{P}\Big[f(\mathbf{X}_{t-1}^{IT}) = 0 \;\Big|\; f(\mathbf{X}_{t-1}^{IT}) \le L-t\Big].$$

By Claim 5.11, we have

$$\mathbf{P}\Big[f(\mathbf{X}_t^{IT}) \le L-t-1 \;\Big|\; 0 < f(\mathbf{X}_{t-1}^{IT}) \le L-t\Big] \ge 1 - \Big[1 - \frac{1}{L}\big(1-\frac{1}{L}\big)^{L-1}\Big]^{L^{1+\epsilon}}$$

$$\ge 1 - \Big[1 - \frac{1}{L}\big(1-\frac{1}{L}\big)^{L}\Big]^{L^{1+\epsilon}}.$$

Conditioned on $f(s_{t-1}^b) = 0$, we have by (5.49) that

$$\mathbf{\Pi}_t(1;i) = 1 - \frac{1}{L} \quad \text{for all } i = 1,\dots,L.$$

Therefore

$$\mathbf{P}\Big[f(\mathbf{X}_t^{IT}) \le L-t-1 \;\Big|\; f(\mathbf{X}_{t-1}^{IT}) = 0\Big] \ge \mathbf{P}\Big[f(\mathbf{X}_t^{IT}) = 0 \;\Big|\; f(\mathbf{X}_{t-1}^{IT}) = 0\Big]$$

$$\ge 1 - \Big[1 - \big(1-\frac{1}{L}\big)^{L}\Big]^{L^{1+\epsilon}}$$

$$\ge 1 - \Big[1 - \frac{1}{L}\big(1-\frac{1}{L}\big)^{L}\Big]^{L^{1+\epsilon}}.$$

Consequently, by (5.54)

$$\mathbf{P}\Big[f(\mathbf{X}_t^{IT}) \le L-t-1 \;\Big|\; f(\mathbf{X}_{t-1}^{IT}) \le L-t\Big] \ge 1 - \Big[1 - \frac{1}{L}\big(1-\frac{1}{L}\big)^{L}\Big]^{L^{1+\epsilon}}. \tag{5.55}$$

Then by (5.51), (5.53) and (5.55), we have

$$\mathbf{P}\big[\forall L-1\geq t\leq 0 \quad f(\mathbf{X}_t^{IT})\leq L-t-1\big]\geq \Big[1-2^{-L^{2+\epsilon}}\Big]\Big[1-\Big[1-\frac{1}{L}\big(1-\frac{1}{L}\big)^L\Big]^{L^{1+\epsilon}}\Big]^L.$$

This in turn implies

$$\mathbf{P}\big[\tau\leq L\big]\geq \Big[1-2^{-L^{2+\epsilon}}\Big]\Big[1-\Big[1-\frac{1}{L}\big(1-\frac{1}{L}\big)^L\Big]^{L^{1+\epsilon}}\Big]^L.$$

As L large enough we have

$$(1-\frac{1}{L})^L\geq \tfrac{1}{2}e^{-1},$$

in turn

$$1-\Big[1-\frac{1}{L}\big(1-\frac{1}{L}\big)^L\Big]^{L^{1+\epsilon}}\geq 1-\Big[1-\frac{e^{-1}}{2\cdot L}\Big]^{L^{1+\epsilon}}=1-\bigg(\Big[1-\frac{e^{-1}}{2\cdot L}\Big]^L\bigg)^{L^\epsilon}.$$

Since

$$\Big[1-\frac{e^{-1}}{2\cdot L}\Big]^L\to e^{-\frac{e^{-1}}{2}}<1 \quad \text{as} \quad L\to\infty,$$

there exists $\beta\in(0,1)$ such that

$$\Big[1-\frac{e^{-1}}{2\cdot L}\Big]^L\leq\beta<1 \quad \text{for } L \text{ large enough.}$$

Therefore, we have

$$1-\Big[1-\frac{1}{L}\big(1-\frac{1}{L}\big)^L\Big]^{L^{1+\epsilon}}\geq 1-\beta^{L^\epsilon} \quad \text{for } L \text{ large enough.}$$

Consequently, for L large enough we have

$$\begin{aligned}
\mathbf{P}\big[\forall L-1\geq t\leq 0 \quad f(\mathbf{X}_t^{IT})\leq L-t-1\big] &\geq \Big[1-2^{-L^{2+\epsilon}}\Big]\Big[1-\beta^{L^\epsilon}\Big]^L\\
&=\Big[1-2^{-L^{2+\epsilon}}\Big]\cdot e^{L\ln(1-\beta^{L^\epsilon})}\\
&=\Big[1-2^{-L^{2+\epsilon}}\Big]\cdot e^{-L\beta^{L^\epsilon}\frac{\ln(1-\beta^{L^\epsilon})}{-\beta^{L^\epsilon}}}\geq\Big[1-2^{-L^{2+\epsilon}}\Big]\cdot e^{-\frac{L\beta^{L^\epsilon}}{2}}\\
&=\Big[1-2^{-L^{2+\epsilon}}\Big]\cdot\Big[1-\big(1-e^{-\frac{L\beta^{L^\epsilon}}{2}}\big)\Big]\\
&\geq\Big[1-2^{-L^{2+\epsilon}}\Big]\cdot\Big[1-\frac{L\beta^{L^\epsilon}}{2}\Big]\\
&=1-\tfrac{L\beta^{L^\epsilon}}{2}-2^{-L^{2+\epsilon}}+-2^{-L^{2+\epsilon}}\ \tfrac{L\beta^{L^\epsilon}}{2}=1-o(\beta^{L^\epsilon}/2).
\end{aligned}$$

Where we use the facts that

$$\frac{\ln(1-\beta^{L^\epsilon})}{-\beta^{L^\epsilon}}\to 1 \quad \text{as } L\to\infty,$$

$$1-e^{-x}\leq x \quad \text{for all } x\geq 0, \quad \text{and } L\beta^{L^\epsilon}\to 0 \quad \text{as } L\to\infty.$$

□

From Theorem 5.8 and 5.9, we know that in the case of a constant learning rate, the key point for reaching an optimal solution quickly is neither to use the best solution in the history to update models, nor to employ a big learning rate. The key point is how to reasonably compensate global exploitation during the search progress. These two Theorems may inspire us that we can compensate global exploration by adapting sample size to the problem size or employing a suitable lower bound to restrict the models.

6 Analysis of model-based search II: absorption of solutions and models

In Chapter 5, we proposed conditions which guarantee finite reachability, and conditions which imply a *stochastically* polynomial runtime. In particular, we see that the popular setting $\rho_t \equiv \rho > 0$ can not guarantee finite reachability i.e. $\mathbf{P}[\tau < \infty] < 1$, but it may make the framework *efficiently* reach an optimal solution with a very high probability (see Theorems 5.8-5.9). To understand this 'peculiar' phenomenon, we need to inspect the underlying stochastic process more closely, especially the sampling process $\left(\mathbf{X}_t\right)_{t=0,1,2,\ldots}$ and models process $\left(\mathbf{\Pi}_t\right)_{t=0,1,2,\ldots}$. We want to see what happens to the sampling process in the case $\rho_t \equiv \rho > 0$.

In Section 6.2, we shall show that in the more general case $\rho_t > \rho > 0$ for each $t \geq 1$, the sampling process almost surely absorbs to a state[1] consisting of an identical solution $s_{<\infty}$ in *finitely* many iterations (i.e. absorption of solutions, see Theorem 6.1). This means that when we set $\rho_t > \rho > 0$ for each $t \geq 1$, the sampling of the framework can preserve *randomness* only in finitely many iterations, after that the sampling becomes deterministic and always produces the same solution $s_{<\infty}$. We shall show also that absorption of solutions actually implies $\mathbf{P}[\tau < \infty] < 1$ (see Theorem 6.4). This further explains why we can not guarantee to reach an optimal solution in the case $\rho_t \equiv \rho > 0$. Moreover, we find that when absorption of solutions holds, the absorbing solution $s_{<\infty}$ is generally a best solution (iteration best or best so far) in the search history (see Theorem 6.3). Recall that the framework learns the empirical models from some selected 'best' solutions. This finding actually shows that with the basic recursion to update models, the next model may effectively catch the good properties in the selected solutions. Actually, absorption of solutions is not a particular property for MBS algorithms. It may also occur in other heuristic search procedures. For example, the famous 'genetic drift' [AM94] in genetic algorithms which describes loss of all variation in solutions.

The intrinsic difference of MBS algorithms with SBS algorithms (i.e. solution-based search algorithms) is that it optimizes models instead of solutions. The asymptotic behavior of the models process $\left(\mathbf{\Pi}_t\right)_{t=0,1,2,\ldots}$ is therefore of great importance for MBS. We inspect the models process in Section 6.3. We will show that when absorption of solutions holds, the models also converge to a limit which concentrates on a single solution s_∞ (i.e. absorption of models, see Theorem 6.5). In particular, when the memories record only some best solutions seen in history, the limiting model may concentrate exactly on an optimal solution (see Theorem 6.6 and Theorem 6.7). However, we are not able to show this for the non-memory case. But, in our former work [WK14b], we have known

[1] Recall that the state space of the sampling process is S^N (N is the sample size), therefore a state is a vector containing N feasible solutions.

that absorption of models and finite reachability are compatible in the non-memory case (see also Theorem 6.8).

The whole Chapter is arranged as: Section 6.1 formally defines the absorption of solutions and absorption of models, then list some additional assumptions; Section 6.2 concentrates on absorption of solutions; Section 6.3 concentrates on absorption of models.

6.1 Definitions and some additional assumptions

This Section shall formally define the two limiting behaviors: absorption of solutions and absorption of models. The proofs in this Chapter may involve some mathematic properties of the memory update rule and subsample selection rule. As a mathematic formalization, we now make some additional assumptions on them. Note that the general assumptions in Subsection 5.1.2 are also assumed throughout this Chapter.

Recall that the underlying CO instance is $(S, f, 0)$ with $S \subseteq \mathcal{A}^L$ for some alphabet $\mathcal{A}$ and strings length $L \in \mathbb{N}$. The models are restricted to $\mathbb{P} = \mathbb{P}_{ce}$, and we use the feasibility construction to generate random solutions.

6.1.1 Definitions

Mathematically, we say that *absorption of solutions* holds in an algorithm if and only if

$$\mathbf{P}\big[\exists s \in S\ \exists T \in \mathbb{N}\ \forall t \geq T\ \mathbf{X}_t = (s, \ldots, s)\big] = 1. \tag{6.1}$$

Obviously, absorption of solutions means that the sample process $(\mathbf{X}_t)_{t\in\mathbb{N}}$ will be almost surely frozen at a single solution after finitely many iterations. Note that in absorption of solutions, the algorithm may gradually narrow its search range on S and eventually keep eyes only on a particular solution. Hence, absorption of solutions may also mean a strong local exploitation. Thereby, for the sake of efficiency, absorption of solutions may be strongly preferred in practice if finite reachability can also be guaranteed. In the sequel, we shall write $s_{<\infty}$ as the *absorbing solution* provided it exists.

We say that an MBS algorithm possesses *absorption of models* if and only if

$$\begin{aligned}\mathbf{P}\big[\exists s \in S\ \exists \mathbf{\Pi}_\infty \in \mathbb{P}\ \forall a \in \mathcal{A}\ \forall i = 1, \ldots, L \\ \mathbf{\Pi}_\infty(s) = 1 \text{ and } \lim_{t\to\infty} \mathbf{\Pi}_t(a; i) = \mathbf{\Pi}_\infty(a; i)\big] = 1.\end{aligned} \tag{6.2}$$

Absorption of models means that the algorithm would gradually concentrate on a particular solution probabilistically. In the sequel, we shall denote the *limit of the models* by $\mathbf{\Pi}_\infty \in \mathbb{P}$ and the finally *concentrated solution* by $s_\infty \in S$ if they do exist.

6.1.2 Some additional assumptions

Let $\mathbf{X}_t^{ITM}$ be the best solution in $\mathbf{X}_t \cup \mathbf{M}_t$, and $\mathbf{X}_t^{IT}$ be the iteration best solution in $\mathbf{X}_t$. Obviously, $f(\mathbf{X}_t^{ITM}) \leq f(\mathbf{X}_t^{IT})$. In the non-memory case, $\mathbf{X}_t^{IT} = \mathbf{X}_t^{ITM}$ (i.e. $\mathbf{M}_t \equiv \emptyset$). And if we use global memory update, $\mathbf{X}_t^{ITM} = \mathbf{X}_t^{BF}$ where recall that $\mathbf{X}_t^{BF}$ is the best solution found among iterations $t = 0, 1, 2, \ldots, t$.

Recall that three commonly used memory update rules are NM (non memory i.e. $\mathbf{M}_t \equiv \emptyset$), GTMU (global truncate memory update, we take M best solutions in $\mathbf{M}_t \cup \mathbf{X}_t$ as the next memory $\mathbf{M}_{t+1}$, see also (4.14)) and LTMU (local truncate memory update, we remove a number of solutions in $\mathbf{M}_t$ first and form $\mathbf{M}_{t+1}$ with the resulting $\mathbf{M}_t$ by adding the same number of best solutions in $\mathbf{X}_t$, see also (4.15)). With NM and GTMU to update memory, we assume that

$$\mathbf{P}\big[\mathbf{X}_t^{ITM} \in \mathcal{N}_t^b\big] \geq \gamma_1 \tag{6.3}$$

for some constant $\gamma_1 \in (0, 1]$ independent of the contents in $\mathbf{M}_t \cup \mathbf{X}_t$, where $\mathbf{P}\big[\mathbf{X}_t^{ITM} \in \mathcal{N}_t^b\big]$ is the probability for the event that the best solution $\mathbf{X}_t^{ITM}$ in $\mathbf{X}_t \cup \mathbf{M}_t$ is chosen to the subsample $\mathcal{N}_t^b$ in iteration t. Therefore, (6.3) actually assumes a time-independent probability lower bound for taking the best solution in $\mathbf{X}_t \cup \mathbf{M}_t$ into $\mathcal{N}_t^b$.

Recall that commonly used subsample selection rules are ID (identity selection, $\mathcal{N}_t^b = \mathbf{X}_t$), TS (truncate selection see p. 60, we take N_b best solutions in $\mathbf{M}_t \cup \mathbf{X}_t$ as $\mathcal{N}_t^b$), RS (random selection, we choose solutions with a distribution based on solutions qualities, see also (4.16)), MRS (memory random selection, random selection in the memory, see also (4.17)) and MID (memory identity selection, $\mathcal{N}_t^b = \mathbf{M}_{t+1}$). With ID or TS as the selection rule, we can set $\gamma_1 = 1$, since $\mathbf{X}_t^{ITM}$ must be in $\mathcal{N}_t^b$ in these two cases.

In other cases, assumption (6.3) is also reasonable. Note that if we do not use memory i.e. $\mathcal{M} =$ NM, the subsample selection rules are generally ID, TS and RS. In the case of $\mathcal{M} =$ NM and $\mathcal{S} =$ RS, due to finiteness of the state space S^N of the samples and positiveness of the weight function g in (4.16), we can set

$$\gamma_1 := \min\Big\{\mathbf{P}\big[s \in \mathcal{N}_t^b \;\big|\; s \in \mathbf{X}_t \cup \mathbf{M}_t = \mathbf{x}\big] \;\Big|\; \mathbf{x} \in S^{N+M}, s \in \mathbf{x}\Big\} > 0,$$

where N is the sample size, M is the memory size which equals 0 here, $\mathbf{P}\big[s \in \mathcal{N}_t^b \;\big|\; s \in \mathbf{X}_t = \mathbf{x}\big]$ is defined as in (4.16) which is always positive under RS. For the case of $\mathcal{M} =$GTMU, we can set γ_1 in a completely identical manner, where observe that $\mathbf{X}_t^{ITM} \in \mathbf{M}_{t+1}$ always holds under GTMU.

In the case of $\mathcal{M} =$ LTMU, memory is maintained by an out rule. Recall that the two most popular out rules are WO (worst out) and FIFO (first in first out). With WO, we kick out a fix number of worst solutions from the memory. Note that in LTMU with WO, assumption (6.3) also holds because $\mathbf{X}_t^{ITM}$ must be taken into the next memory $\mathbf{M}_{t+1}$. Comparatively, in LTMU with FIFO, $\mathbf{X}_t^{ITM}$ may be not in $\mathbf{M}_{t+1}$, but $\mathbf{X}_t^{IT}$ always enters $\mathbf{M}_{t+1}$. And in this case, we can assume that

$$\mathbf{P}\big[\mathbf{X}_t^{IT} \in \mathcal{N}_t^b\big] \geq \gamma_2 \tag{6.4}$$

for a constant $\gamma_2 \in (0, 1]$ also independent of the contents in $\mathbf{M}_t \cup \mathbf{X}_t$. Practically, we often combine LTMU with RS or ID selection, e.g. PBAS (FIFO+ID or WO+ID) and $\mathcal{MMAS}$ (WO+RS). A similar discussion as in the case of NM and GTMU, we can see that assumption (6.4) is also reasonable.

6.2 Absorption of solutions in model-based search

In genetic algorithms, the phenomenon of *genetic drift* is well-known, see e.g.[AM94]. It describes the loss of all variation in the produced solutions by a crossover. Genetic drift may result in a *premature stagnation* i.e. the qualities of solutions stop improving before seeing an optimal solution. In this section, we show that a similar phenomenon may also happen in MBS. In particular, we show that the marginal process $(\mathbf{X}_t)_{t\in\mathbb{N}}$ may be absorbed into a single solution in finitely many iterations, and optimal solutions may not be seen in this case.

6.2.1 A universal absorption Theorem

Theorem 6.1 below extends a finding in our former work [WK14b] and [WK14a]. In [WK14b], we showed that absorption of solutions holds in CE and AS for the case of a constant learning rate. And in [WK14a], we showed that absorption of solutions also holds if we do not use memory i.e. the case $\mathcal{M} = \text{NM}$. In these two work, we employed a rather complex proof. Here, we shall use a different and much easier proof to show that absorption of solutions also holds for other memory update rules in the framework.

The Theorem depends on the two additional assumptions (6.3) and (6.4). As we discussed in Subsection 6.1.2, these assumptions do not impose any restriction on the underlying memory update and subsample selection. They are actually mathematical formalizations. Therefore, the Theorem can apply to all MBS algorithms covered by our framework. So, it is a universal absorption theorem or drift theory for model-based search.

Theorem 6.1. *Assume $\rho_t \geq \rho > 0$ for all $t \in \mathbb{N}$ and a constant $\rho \in (0, 1]$, $N \in \mathbb{N}$ is a constant sample size and $M \in \mathbb{N}$ is a constant memory size. Then the following hold.*

a) *Assume* (6.3) *and we do not use memory, then absorption of solutions holds.*

b) *Assume* (6.3) *and $\mathcal{M} = GTMU$, then absorption of solutions holds.*

c) *Assume* (6.3) *and $\mathcal{M} = LTMU$ with WO, then absorption of solutions holds.*

d) *Assume* (6.4) *and $\mathcal{M} = LTMU$ with FIFO, then absorption of solutions holds*

Proof. By definition (6.1), absorption of solutions means

$$\mathbf{P}\big[\text{exists } s \in S \text{ exists } T \in \mathbb{N} \text{ for all } t \geq T \quad X_t = (s, \dots, s)\big] = 1.$$

Let $M_1 := \min\{t \in \mathbb{N} \mid \exists s, s' \in \mathbf{X}_t \cup \mathbf{X}_{t+1} \quad s \neq s'\}$. For each $k > 1$, let $M_k := \min\{t > M_{k-1} \mid \exists s, s' \in \mathbf{X}_t \cup \mathbf{X}_{t+1} \quad s \neq s'\}$. Here, we use a convention that $\min \emptyset = \infty$. Obviously $M_k \geq k - 1$ and

$$M_{k+1} < \infty \;\Rightarrow\; M_k < \infty \quad \text{for all } k \geq 1.$$

Let $d = 1, 2, \dots$ be an arbitrarily fixed integer. For any $T \in \mathbb{N}$ and $s \in S$, the event [for all $t \geq T \quad X_t = (s, \dots, s)$] implies $[M_{d\cdot(T+1)} = \infty]$. So, absorption of solutions implies that for any integer $d = 1, 2, 3, \dots$,

$$\mathbf{P}\big[\text{exists } k \geq 1 \quad M_{d\cdot k} = \infty\big] = 1. \tag{6.5}$$

Note that if (6.5) holds, then almost surely that

$$\text{for all } t \geq M_{d\cdot k_0-1}+1, s, s' \in \mathbf{X}_t \cup \mathbf{X}_{t+1} \quad s = s' \tag{6.6}$$

where $k_0 := \min\{k \in \mathbb{N} \mid M_{d\cdot k} = \infty\}$. Obviously, event (6.6) implies [exists $s \in S$ exists $T \in \mathbb{N}$ for all $t \geq T \quad X_t = (s, \ldots, s)$]. Hence (6.5) again implies absorption of solutions. Therefore, absorption of solutions is equivalent to (6.5), for any $d = 1, 2, \ldots$.

Now, to prove absorption of solutions under different memory update rules, we just need to show (6.5) holds resp., for some constant $d \geq 1$. To achieve this, we first show a Lemma below which states an achievable sufficient condition for (6.5).

Lemma 6.2. *Given a constant integer $d \geq 1$. If there exists a constant $\kappa \in (0, 1]$ such that for any $k \geq 1$ and any $m \geq d \cdot k - 1$*

$$\mathbf{P}\big[M_{d\cdot(k+1)} = \infty \;\big|\; M_{d\cdot k} = m < \infty\big] \geq \kappa > 0, \tag{6.7}$$

then (6.5) *holds.*

Proof. Note that

$$\begin{aligned}
&\mathbf{P}\big[\forall k \geq 1 \quad M_{d\cdot k} < \infty\big] \\
&\quad = \mathbf{P}[M_d < \infty] \prod_{k=1}^{\infty} \mathbf{P}\big[M_{d\cdot(k+1)} < \infty \;\big|\; M_{d\cdot l} < \infty,\ \forall 1 \leq l \leq k\big] \\
&\quad = \mathbf{P}[M_d < \infty] \prod_{k=1}^{\infty} \mathbf{P}\big[M_{d\cdot(k+1)} < \infty \;\big|\; M_{d\cdot k} < \infty\big] \\
&\quad = \mathbf{P}[M_d < \infty] \prod_{k=1}^{\infty} \Big[1 - \mathbf{P}\big[M_{d\cdot(k+1)} = \infty \;\big|\; M_{d\cdot k} < \infty\big]\Big]
\end{aligned}$$

where we use the fact that

$$M_{d\cdot k_1} < \infty \Rightarrow M_{d\cdot k_2} < \infty, \quad \text{if } k_1 > k_2.$$

Note also that

$$\begin{aligned}
&\mathbf{P}\big[M_{d\cdot(k+1)} = \infty \;\big|\; M_{d\cdot k} < \infty\big] \\
&\qquad = \sum_{m \geq d\cdot k-1} \mathbf{P}\big[M_{d\cdot(k+1)} \;\; = \infty \;\big|\; M_{d\cdot k} = m < \infty\big]\mathbf{P}\big[\, M_{d\cdot k} = m \;\big|\; M_{d\cdot k} < \infty\big]
\end{aligned}$$

and

$$\sum_{m \geq d\cdot k-1} \mathbf{P}\big[\, M_{d\cdot k} = m \;\big|\; M_{d\cdot k} < \infty\big] = 1.$$

Therefore, with assumption (6.7) we have

$$\mathbf{P}\big[M_{d\cdot(k+1)} = \infty \;\big|\; M_{d\cdot k} < \infty\big] \geq \kappa > 0 \quad \text{for all } k \geq 1.$$

Which in turn implies $\mathbf{P}\big[\forall k \geq 1 \quad M_{d\cdot k} < \infty\big] = 0$ and (6.5). □

Now, we show (6.5) through proving (6.7) under different memory update rules which in turn implies statements a)-d).

a) (see [WK14a] for an alternative proof) Assume (6.3) and $\mathbf{M}_t \equiv \emptyset$. To prove (6.7), we take $d = 2$ and arbitrarily fix a $k \geq 1$ and an $m \geq 2k-1$. Now, we are going to look for a lower bound $\kappa > 0$ for (6.7).

Note that the random event

$$[\forall t \geq m+2 \quad \mathbf{X}_t = (\mathbf{X}_{m+1}^{ITM}, \ldots, \mathbf{X}_{m+1}^{ITM})]$$

implies $[M_{2k+2} = \infty]$ conditioned on $[M_{2k} = m < \infty]$, where recall that $\mathbf{X}_{m+1}^{ITM}$ is the best solution in $\mathbf{X}_{m+1} \cup \mathbf{M}_{m+1}$ and it coincides with $\mathbf{X}_{m+1}^{IT}$ since $\mathbf{M}_{m+1} = \emptyset$ here. So, we have

$$\begin{aligned}\mathbf{P}\big[M_{2\cdot(k+1)} = \infty \;\big|\; M_{2\cdot k} = m < \infty\big] \\ \geq \mathbf{P}\big[\forall t \geq m+2 \quad \mathbf{X}_t = (\mathbf{X}_{m+1}^{ITM}, \ldots, \mathbf{X}_{m+1}^{ITM}) \;\big|\; M_{2\cdot k} = m < \infty\big].\end{aligned}$$

Hence, to show (6.7), we just need to find a positive lower bound κ for

$$\mathbf{P}\big[\forall t \geq m+2 \quad \mathbf{X}_t = (\mathbf{X}_{m+1}^{ITM}, \ldots, \mathbf{X}_{m+1}^{ITM}) \;\big|\; M_{2\cdot k} = m < \infty\big].$$

Now we abbreviate $[\mathbf{X}_t = (\mathbf{X}_{m+1}^{ITM}, \ldots, \mathbf{X}_{m+1}^{ITM})]$ as $[\mathbf{X}_t \equiv \mathbf{X}_{m+1}^{ITM}]$. Then

$$\begin{aligned}&\mathbf{P}\big[\forall t \geq m+2 \quad \mathbf{X}_t = (\mathbf{X}_{m+1}^{ITM}, \ldots, \mathbf{X}_{m+1}^{ITM}) \;\big|\; M_{2\cdot k} = m < \infty\big] \\ &\quad = \mathbf{P}\big[\forall t \geq m+2 \quad \mathbf{X}_t \equiv \mathbf{X}_{m+1}^{ITM} \;\big|\; M_{2\cdot k} = m < \infty\big] \\ &\quad = \mathbf{P}\big[\mathbf{X}_{m+2} \equiv \mathbf{X}_{m+1}^{ITM} \;\big|\; M_{2\cdot k} = m < \infty\big] \prod_{t=m+3}^{\infty} \mathbf{P}\big[\mathbf{X}_t \equiv \mathbf{X}_{m+1}^{ITM} \\ &\qquad\qquad \;\big|\; M_{2\cdot k} = m < \infty, \forall t-1 \geq l \geq m+2 \quad \mathbf{X}_l \equiv \mathbf{X}_{m+1}^{ITM}\big].\end{aligned} \tag{6.8}$$

By assumption (6.3), we know that

$$\mathbf{P}[\mathbf{X}_{m+1}^{ITM} \in \mathcal{N}_{m+1}^{b}] > \gamma_1 > 0.$$

Since γ_1 is independent of the contents in the sample, by the general assumption of learning rule (5.8) and (5.10), we have

$$\begin{aligned}&\mathbf{P}\big[\forall L \geq i \geq 1 \quad \mathbf{W}_{m+1}\big(\mathbf{X}_{m+1}^{ITM}(i); i\big) \geq \alpha \;\big|\; M_{2\cdot k} = m < \infty\big] \\ &\quad = \mathbf{P}\big[\forall L \geq i \geq 1 \quad \mathbf{W}_{m+1}\big(\mathbf{X}_{m+1}^{ITM}(i); i\big) \geq \alpha \;\big|\; \mathbf{X}_{m+1}^{ITM} \in \mathcal{N}_{m+1}^{b}, M_{2\cdot k} = m < \infty\big] \\ &\qquad\qquad \cdot \mathbf{P}\big[\mathbf{X}_{m+1}^{ITM} \in \mathcal{N}_{m+1}^{b} \;\big|\; M_{2\cdot k} = m < \infty\big] \\ &\quad = \mathbf{P}\big[\mathbf{X}_{m+1}^{ITM} \in \mathcal{N}_{m+1}^{b} \;\big|\; M_{2\cdot k} = m < \infty\big] \geq \gamma_1 > 0\end{aligned} \tag{6.9}$$

where we write $\mathbf{X}_{m+1}^{ITM}$ as $\big(\mathbf{X}_{m+1}^{ITM}(1), \ldots, \mathbf{X}_{m+1}^{ITM}(L)\big)$. Then by (6.9), the first factor in (6.8) can be written as

$$\begin{aligned}&\mathbf{P}\big[\mathbf{X}_{m+2} \equiv \mathbf{X}_{m+1}^{ITM} \;\big|\; M_{2\cdot k} = m < \infty\big] \\ &\quad = \mathbf{P}\big[\mathbf{X}_{m+2} \equiv \mathbf{X}_{m+1}^{ITM} \;\big|\; M_{2\cdot k} = m < \infty, \forall L \geq i \geq 1 \quad \mathbf{W}_{m+1}\big(\mathbf{X}_{m+1}^{ITM}(i); i\big) \geq \alpha\big] \\ &\qquad\qquad \cdot \mathbf{P}\big[\forall L \geq i \geq 1 \quad \mathbf{W}_{m+1}\big(\mathbf{X}_{m+1}^{ITM}(i); i\big) \geq \alpha \;\big|\; M_{2\cdot k} = m < \infty\big] \\ &\quad \geq \gamma_1 \cdot \mathbf{P}\big[\mathbf{X}_{m+2} \equiv \mathbf{X}_{m+1}^{ITM} \;\big|\; M_{2\cdot k} = m < \infty, \forall L \geq i \geq 1 \quad \mathbf{W}_{m+1}\big(\mathbf{X}_{m+1}^{ITM}(i); i\big) \geq \alpha\big].\end{aligned} \tag{6.10}$$

Note that

$$\begin{aligned}\mathbf{P}\big[\mathbf{X}_{m+2} \equiv \mathbf{X}_{m+1}^{ITM} \;\big|\; M_{2\cdot k} = m < \infty, \forall L \geq i \geq 1 \quad \mathbf{W}_{m+1}\big(\mathbf{X}_{m+1}^{ITM}(i); i\big) \geq \alpha\big] \\ = \mathbf{E}\Big[\big(Q(\mathbf{X}_{m+1}^{ITM}; \mathbf{\Pi}_{m+2})\big)^N \;\Big|\; M_{2\cdot k} = m < \infty, \forall L \geq i \geq 1 \quad \mathbf{W}_{m+1}\big(\mathbf{X}_{m+1}^{ITM}(i); i\big) \geq \alpha\Big],\end{aligned} \tag{6.11}$$

and by definition (5.16)

$$\begin{aligned}Q(\mathbf{X}_{m+1}^{ITM}; \mathbf{\Pi}_{m+2}) &= \prod_{i=1}^{L} Q\big(\mathbf{X}_{m+1}^{ITM}(i); \mathbf{X}_{m+1\,|i-1}^{ITM}, i, \mathbf{\Pi}_{m+2}\big) \\ &\geq \prod_{i=1}^{L} C_{i-1}\big(\mathbf{X}_{m+1\,|i-1}^{ITM}; \mathbf{X}_{m+1}^{ITM}(i)\big)\mathbf{\Pi}_{m+2}\big(\mathbf{X}_{m+1}^{ITM}(i); i\big) \\ &\geq \lambda^L \prod_{i=1}^{L} \mathbf{\Pi}_{m+2}\big(\mathbf{X}_{m+1}^{ITM}(i); i\big),\end{aligned}$$

where $\lambda > 0$ is defined in (5.20). By the basic recursion (4.21), have that

$$\mathbf{\Pi}_{m+2}\big(\mathbf{X}_{m+1}^{ITM}(i); i\big) \geq \rho_{m+2} \cdot \alpha \geq \rho \cdot \alpha \quad \text{for all } i = 1, \ldots, L$$

conditioned on the event

$$[\forall L \geq i \geq 1 \quad \mathbf{W}_{m+1}\big(\mathbf{X}_{m+1}^{ITM}(i); i\big) \geq \alpha].$$

Therefore (6.11) can be bounded as

$$\begin{aligned}&\mathbf{E}\Big[\big(Q(\mathbf{X}_{m+1}^{ITM}; \mathbf{\Pi}_{m+2})\big)^N \;\Big|\; M_{d\cdot k} = m < \infty, \forall L \geq i \geq 1 \quad \mathbf{W}_{m+1}\big(\mathbf{X}_{m+1}^{ITM}(i); i\big) \geq \alpha\Big] \\ &\geq \mathbf{E}\Big[\big(\lambda \cdot \alpha \cdot \rho\big)^{L \cdot N} \;\Big|\; M_{d\cdot k} = m < \infty, \forall L \geq i \geq 1 \quad \mathbf{W}_{m+1}\big(\mathbf{X}_{m+1}^{ITM}(i); i\big) \geq \alpha\Big] \\ &= \big(\lambda \cdot \alpha \cdot \rho\big)^{L \cdot N} > 0.\end{aligned} \tag{6.12}$$

As a result of (6.12) and (6.10), the first factor in (6.8) can be bounded as

$$\mathbf{P}\big[\mathbf{X}_{m+2} \equiv \mathbf{X}_{m+1}^{ITM} \;\big|\; M_{d\cdot k} = m < \infty\big] \geq \gamma_1 \cdot \big(\lambda \cdot \alpha \cdot \rho\big)^{L \cdot N} > 0 \tag{6.13}$$

which is independent of k and m.

Now we are going to analyze the lower bound for the infinite products in (6.8). Note that for each $t \geq m + 3$, due to the Markov property (4.22) of the underlying process $\big(\mathbf{\Pi}_t, \mathbf{X}_t, \mathbf{M}_t, \mathcal{N}_t^b, \mathbf{W}_t\big)_{t=0,1,2,\ldots}$ we have

$$\begin{aligned}\mathbf{P}\big[\mathbf{X}_t \equiv \mathbf{X}_{m+1}^{ITM} \;\big|\; M_{2\cdot k} = m < \infty, \forall t-1 \geq l \geq m+2 \quad \mathbf{X}_l \equiv \mathbf{X}_{m+1}^{ITM}\big] \\ = \mathbf{P}\big[\mathbf{X}_t \equiv \mathbf{X}_{m+1}^{ITM} \;\big|\; \forall t-1 \geq l \geq m+2 \quad \mathbf{X}_l \equiv \mathbf{X}_{m+1}^{ITM}\big].\end{aligned}$$

Since we do not use memory here, conditioned on the event

$$[\forall t-1 \geq l \geq m+2 \quad \mathbf{X}_l \equiv \mathbf{X}_{m+1}^{ITM}] \tag{6.14}$$

we have

$$\Big[\forall t-1 \geq l \geq m+2 \quad \mathcal{N}_l^b = \big(\mathbf{X}_{m+1}^{ITM}, \ldots, \mathbf{X}_{m+1}^{ITM}\big)\Big].$$

By assumption (5.9), we have

$$\Big[\forall t-1 \geq l \geq m+2 \; \forall i = 1, \ldots, L \quad \mathbf{W}_l\big(\mathbf{X}_{m+1}^{ITM}(i); i\big) = 1\Big].$$

Consequently (5.23) in Lemma 5.6 holds for $y := \mathbf{X}_{m+1\,|i-1}^{ITM}$ and $T := m+2$ for all $i = 1, 2, \ldots, L$. Hence, conditioned on (6.14), we may apply $\rho_l \geq \rho > 0$ and (5.26) of Lemma 5.6 with $w := 1$ to get that

$$\begin{aligned} Q'\big(\mathbf{X}_{m+1}^{ITM}; \mathbf{X}_{m+1\,|i-1}^{ITM}, i, \mathbf{\Pi}_t\big) &\geq 1 - \Big(1 - Q'\big(\mathbf{X}_{m+1}^{ITM}; \mathbf{X}_{m+1\,|i-1}^{ITM}, i, \mathbf{\Pi}_{m+2}\big)\Big) \prod_{l=m+3}^{t} (1-\rho_l) \\ &\geq 1 - \Big(1 - Q'\big(\mathbf{X}_{m+1}^{ITM}; \mathbf{X}_{m+1\,|i-1}^{ITM}, i, \mathbf{\Pi}_{m+2}\big)\Big) \prod_{l=m+3}^{t} (1-\rho) \\ &\geq 1 - (1-\rho)^{t-m-2}, \end{aligned} \tag{6.15}$$

for all $i = 1, 2, \ldots, L$. By (6.15) and a) of Lemma 5.5, we have

$$\begin{aligned} Q(\mathbf{X}_{m+1}^{ITM}; \mathbf{\Pi}_t) &= \prod_{i=1}^{L} Q\big(\mathbf{X}_{m+1}^{ITM}; \mathbf{X}_{m+1\,|i-1}^{ITM}, i, \mathbf{\Pi}_t\big) \\ &\geq \prod_{i=1}^{L} \ell\Big[Q\big(\mathbf{X}_{m+1}^{ITM}; \mathbf{X}_{m+1\,|i-1}^{ITM}, i, \mathbf{\Pi}_t\big)\Big] \\ &\geq \prod_{i=1}^{L} \ell\Big[1 - (1-\rho)^{t-m-2}\Big] \\ &= \Big[\ell\big(1 - (1-\rho)^{t-m-2}\big)\Big]^L. \end{aligned} \tag{6.16}$$

With (6.16) we can now bound each factor in the infinite products of (6.8) as

$$\begin{aligned} &\mathbf{P}\big[\mathbf{X}_t \equiv \mathbf{X}_{m+1}^{ITM} \;\big|\; M_{2\cdot k} = m < \infty, \forall t-1 \geq l \geq m+2 \quad \mathbf{X}_l \equiv \mathbf{X}_{m+1}^{ITM}\big] \\ &\qquad = \mathbf{E}\Big[Q(\mathbf{X}_{m+1}^{ITM}; \mathbf{\Pi}_t)^N \;\Big|\; M_{2\cdot k} = m < \infty, \forall t-1 \geq l \geq m+2 \quad \mathbf{X}_l \equiv \mathbf{X}_{m+1}^{ITM}\Big] \\ &\qquad \geq \Big[\ell\big(1 - (1-\rho)^{t-m-2}\big)\Big]^{L\cdot N}. \end{aligned} \tag{6.17}$$

Combine (6.13) and (6.17), we can bound (6.8) as

$$\begin{aligned}\mathbf{P}\big[\forall t \geq m+2 \quad \mathbf{X}_t = (\mathbf{X}_{m+1}^{ITM}, \ldots, \mathbf{X}_{m+1}^{ITM}) \;\big|\; M_{2\cdot k} = m < \infty\big] \\ \geq \gamma_1 \cdot \big(\lambda \cdot \alpha \cdot \rho\big)^{L\cdot N} \cdot \prod_{t=m+3}^{\infty} \Big[\ell\big(1-(1-\rho)^{t-m-2}\big)\Big]^{L\cdot N} \\ \geq \gamma_1 \cdot \big(\lambda \cdot \alpha \cdot \rho\big)^{L\cdot N} \cdot \Big[\prod_{l=1} \Big(1-\hbar\big((1-\rho)^l\big)\Big)\Big]^{L\cdot N} := \kappa > 0.\end{aligned} \tag{6.18}$$

Where to justify $\kappa > 0$, we use the Lemma 5.4 c). Obviously, κ is independent of k and m. Therefore (6.7) holds for non memory case, so absorption of solutions holds.

b) To show (6.7) holds for GTMU, we take $d = M+1$ and arbitrarily fix a $k \in \mathbb{N}$ and a $m \geq (M+1)\cdot k - 1$, where recall that M is the size of the memory. The proof is similar with the proof for a), we just sketch it.

Similar with the non-memory case, we have

$$\mathbf{P}\big[\forall t \geq m+M+1 \quad \mathbf{X}_t \cup \mathbf{M}_t = (\mathbf{X}_{m+1}^{ITM}, \ldots, \mathbf{X}_{m+1}^{ITM}) \;\big|\; M_{d\cdot k} = m < \infty\big] \geq \kappa > 0$$

implies (6.7).

Note that with GTMU, $\mathbf{X}_{m+1}^{ITM}$ is the best so far solution within iterations $0, 1, \ldots, m+1$. Therefore,

$$\forall m+M+1 \geq t \geq m+2 \quad \mathbf{X}_t \equiv \mathbf{X}_{m+1}^{ITM}$$

implies

$$\mathbf{M}_{m+M+1} \cup \mathbf{X}_{m+M+1} = (\mathbf{X}_{m+1}^{ITM}, \ldots, \mathbf{X}_{m+1}^{ITM}).$$

And observe that

$$\begin{aligned}\mathbf{P}\big[\forall t \geq m+M+1 \quad \mathbf{X}_t \cup \mathbf{M}_t = (\mathbf{X}_{m+1}^{ITM}, \ldots, \mathbf{X}_{m+1}^{ITM}) \;\big|\; M_{d\cdot k} = m < \infty\big] \\ = \mathbf{P}\big[\mathbf{M}_{m+M+1} \cup \mathbf{X}_{m+M+1} \equiv \mathbf{X}_{m+1}^{ITM} \;\big|\; M_{d\cdot k} = m < \infty\big] \prod_{t=m+M+2}^{\infty} \mathbf{P}\Big[\mathbf{M}_t \cup \mathbf{X}_t \equiv \mathbf{X}_{m+1}^{ITM} \;\Big|\; \\ \forall t-1 \geq l \geq m+M+1 \quad \mathbf{X}_l \cup \mathbf{M}_l \equiv \mathbf{X}_{m+1}^{ITM}, M_{d\cdot k} = m < \infty\Big]\end{aligned} \tag{6.19}$$

where we abbreviate $[\mathbf{X}_t \cup \mathbf{M}_t = (\mathbf{X}_{m+1}^{ITM}, \ldots, \mathbf{X}_{m+1}^{ITM})]$ as $[\mathbf{X}_t \cup \mathbf{M}_t \equiv \mathbf{X}_{m+1}^{ITM}]$.

Note that

$$\begin{aligned}\mathbf{P}\big[\mathbf{M}_{m+M+1} \cup \mathbf{X}_{m+M+1} \equiv \mathbf{X}_{m+1}^{ITM} \;\big|\; M_{d\cdot k} = m < \infty\big] \\ \geq \mathbf{P}\big[\forall m+M+1 \geq t \geq m+2 \quad \mathbf{X}_t \equiv \mathbf{X}_{m+1}^{ITM} \;\big|\; M_{d\cdot k} = m < \infty\big] \\ = \mathbf{P}\big[\mathbf{X}_{m+2} \equiv \mathbf{X}_{m+1}^{ITM} \;\big|\; M_{d\cdot k} = m < \infty\big] \prod_{l=m+3}^{m+M+1} \mathbf{P}\Big[\mathbf{X}_l \equiv \mathbf{X}_{m+1}^{ITM} \;\Big|\; \\ \forall l \geq l' \geq m+2 \quad \mathbf{X}_{l'} \equiv \mathbf{X}_{m+1}^{ITM}, M_{d\cdot k} = m < \infty\Big].\end{aligned} \tag{6.20}$$

With completely similar steps as (6.9)-(6.13), we have that

$$\mathbf{P}\big[\mathbf{X}_{m+2} \equiv \mathbf{X}_{m+1}^{ITM} \;\big|\; M_{d\cdot k} = m < \infty\big] \geq \gamma_1 \cdot \big(\lambda \cdot \alpha \cdot \rho\big)^{L\cdot N} > 0.$$

Conditioned on

$$\big[\forall l \geq l' \geq m+2 \quad \mathbf{X}_{l'} \equiv \mathbf{X}^{ITM}_{m+1}\big]$$

we have

$$\mathbf{X}^{ITM}_{m+1} = \mathbf{X}^{ITM}_{m+2} = \cdots = \mathbf{X}^{ITM}_{m+l}.$$

Then with completely similar steps as (6.9)-(6.13), we can also derive that

$$\mathbf{P}\Big[\mathbf{X}_l \equiv \mathbf{X}^{ITM}_{m+1} \;\Big|\; \forall l \geq l' \geq m+2 \quad \mathbf{X}_{l'} \equiv \mathbf{X}^{ITM}_{m+1}, M_{d\cdot k} = m < \infty\Big] \geq \gamma_1 \cdot \big(\lambda \cdot \alpha \cdot \rho\big)^{L\cdot N} > 0.$$

Consequently, by (6.20) we have

$$\mathbf{P}\big[\mathbf{M}_{m+M+1} \cup \mathbf{X}_{m+M+1} \equiv \mathbf{X}^{ITM}_{m+1} \mid M_{d\cdot k} = m < \infty\big] \geq \big[\gamma_1 \cdot \big(\lambda \cdot \alpha \cdot \rho\big)^{L\cdot N}\big]^M > 0. \quad (6.21)$$

Note that conditioned on

$$\big[\forall t-1 \geq l \geq m+M+1 \quad \mathbf{X}_l \cup \mathbf{M}_l \equiv \mathbf{X}^{ITM}_{m+1}\big]$$

event $[\mathbf{X}_t \cup \mathbf{M}_t \equiv \mathbf{X}^{ITM}_{m+1}]$ is equivalent to $[\mathbf{X}_t \equiv \mathbf{X}^{ITM}_{m+1}]$. Therefore, similar steps as in (6.14)-(6.17) show that

$$\mathbf{P}\Big[\mathbf{M}_t \cup \mathbf{X}_t \equiv \mathbf{X}^{ITM}_{m+1} \;\Big|\; \forall t-1 \geq l \geq m+M+1 \quad \mathbf{X}_l \cup \mathbf{M}_l \equiv \mathbf{X}^{ITM}_{m+1}, M_{d\cdot k} = m < \infty\Big] \geq \Big[\ell\big(1-(1-\rho)^{t-M-1}\big)\Big]^{L\cdot N}. \quad (6.22)$$

By (6.21) and (6.22), we have

$$\begin{aligned}\mathbf{P}\big[\forall t \geq m+M+1 \quad \mathbf{X}_t \cup \mathbf{M}_t = (\mathbf{X}^{ITM}_{m+1}, \ldots, \mathbf{X}^{ITM}_{m+1}) \mid M_{d\cdot k} = m < \infty\big] \quad (6.23)\\ \geq \big[\gamma_1 \cdot \big(\lambda \cdot \alpha \cdot \rho\big)^{L\cdot N}\big]^M \cdot \prod_{l=1}^{\infty}\Big(1-\hbar\big((1-\rho)^l\big)\Big)^{L\cdot N} := \kappa > 0\end{aligned}$$

which implies (6.7) for GTMU.

c) Observe that in LTMU with WO, the best solution $\mathbf{X}^{ITM}_t$ in $\mathbf{M}_t \cup \mathbf{X}_t$ is also always taken into $\mathbf{M}_{t+1}$ for each $t \in \mathbb{N}$. We can also take $d = M+1$, and use $\mathbf{X}^{ITM}_{m+1}$ to construct the lower bound. The proof is completely the same as in b).

d) For this case, we take $d = M+1$ also. And for any arbitrarily fixed k and $m \geq (M+1)k - 1$,

$$\mathbf{P}\big[\forall t \geq m+M+1 \quad \mathbf{X}_t \cup \mathbf{M}_t \equiv \mathbf{X}^{IT}_{m+1} \mid M_{d\cdot k} = m < \infty\big] \geq \kappa > 0$$

implies (6.7), where recall that $\mathbf{X}^{IT}_{m+1}$ is the best solution in $\mathbf{X}_{m+1}$.

Similar with (6.19), we have

$$\begin{aligned}\mathbf{P}\big[\forall t \geq m+M+1 \quad \mathbf{X}_t \cup \mathbf{M}_t \equiv \mathbf{X}^{IT}_{m+1} \mid M_{d\cdot k} = m < \infty\big] \quad (6.24)\\ = \mathbf{P}\big[\mathbf{M}_{m+M+1} \cup \mathbf{X}_{m+M+1} \equiv \mathbf{X}^{IT}_{m+1} \mid M_{d\cdot k} = m < \infty\big] \prod_{t=m+M+2}^{\infty} \mathbf{P}\Big[\mathbf{M}_t \cup \mathbf{X}_t \equiv \mathbf{X}^{IT}_{m+1} \;\Big|\\ \forall t-1 \geq l \geq m+M+1 \quad \mathbf{X}_l \cup \mathbf{M}_l \equiv \mathbf{X}^{IT}_{m+1}, M_{d\cdot k} = m < \infty\Big].\end{aligned}$$

Observe that with LTMU,

$$\forall m+M+1 \geq t \geq m+2 \quad \mathbf{X}_t \equiv \mathbf{X}_{m+1}^{IT}$$

implies

$$\mathbf{M}_{m+M+1} \cup \mathbf{X}_{m+M+1} = (\mathbf{X}_{m+1}^{IT}, \ldots, \mathbf{X}_{m+1}^{IT})$$

since conditioned on

$$\forall m+M+1 \geq t \geq m+2 \quad \mathbf{X}_t \equiv \mathbf{X}_{m+1}^{IT}$$

holds

$$\mathbf{X}_{m+1}^{IT} = \mathbf{X}_{m+2}^{IT} = \cdots = \mathbf{X}_{m+M+1}^{IT},$$

and we use FIFO out rule. With similar steps as (6.9)-(6.13), γ_2 instead of γ_1 and assumption (6.4) instead of (6.3), we have that

$$\mathbf{P}\big[\mathbf{M}_{m+M+1} \cup \mathbf{X}_{m+M+1} \equiv \mathbf{X}_{m+1}^{IT} \;\big|\; M_{d\cdot k} = m < \infty\big] \geq \big[\gamma_2 \cdot \big(\lambda \cdot \alpha \cdot \rho\big)^{L\cdot N}\big]^M > 0. \quad (6.25)$$

Similar as in **b)** or in steps (6.14)-(6.17),

$$\mathbf{P}\Big[\mathbf{M}_t \cup \mathbf{X}_t \equiv \mathbf{X}_{m+1}^{IT} \;\Big|\; \forall t-1 \geq l \geq m+M+1 \quad \mathbf{X}_l \cup \mathbf{M}_l \equiv \mathbf{X}_{m+1}^{IT}, M_{d\cdot k} = m < \infty\Big] \geq \Big[\ell\big(1-(1-\rho)^{t-M-1}\big)\Big]^{L\cdot N}. \quad (6.26)$$

By (6.24), (6.25) and (6.26), we have

$$\mathbf{P}\big[\forall t \geq m+M+1 \quad \mathbf{X}_t \cup \mathbf{M}_t \equiv \mathbf{X}_{m+1}^{IT} \;\big|\; M_{d\cdot k} = m < \infty\big] \quad (6.27)$$
$$\geq \big[\gamma_2 \cdot \big(\lambda \cdot \alpha \cdot \rho\big)^{L\cdot N}\big]^M \cdot \prod_{l=1}^{\infty}\Big(1-\hbar\big((1-\rho)^l\big)\Big)\Big]^{L\cdot N} := \kappa > 0$$

which implies (6.7) for LTMU with FIFO. □

6.2.2 A general method for time inhomogeneous Markov chain

Technically, Theorem 6.1 states that the process $\mathbf{X}_t$ absorbs into a state (=sample) $(s, \ldots, s)$ in the finite state space S^N almost surely. It extends the well-known results from time-homogeneous Markov chains as they are used to model drift in simple genetic algorithms to the inhomogeneous case we are considering here.

Let $(U_t)_{t=0,1,2,\ldots}$ be a Markov chain with state space $\mathcal{U}$. And let $\mathcal{Q} = \{\mathcal{Q}(\delta) \mid \delta \in \mathfrak{S}\}$ be a family of statements and $\mathfrak{S}$ is the index set. Then, we say that the chain $(U_t)_{t=0,1,2,\ldots}$ satisfies *tail property* $\mathcal{Q}$ if and only if

$$\mathbf{P}\big[\exists \delta \in \mathfrak{S}\ \exists T \in\ N\ \forall t \geq T \quad U_t \models \mathcal{Q}(\delta)\big] = 1,$$

where notation $U_t \models \mathcal{Q}(\delta)$ means that U_t makes statement $\mathcal{Q}(\delta)$ hold. Obviously, absorption of solutions is a particular tail property of the underlying stochastic process

$$\big(\mathbf{\Pi}_t, \mathbf{X}_t, \mathbf{M}_t, \mathcal{N}_t, \mathbf{W}_t\big)_{t=0,1,2,\ldots}$$

provided we define that

- $\mathcal{Q} = \{\mathcal{Q}(s) \mid s \in S\}$,
- $\big(\mathbf{\Pi}_t, \mathbf{X}_t, \mathbf{M}_t, \mathcal{N}_t, \mathbf{W}_t\big) \models \mathcal{Q}(s)$ if and only if $X_t = (s, \ldots, s)$.

The proof of Theorem 6.1 actually presents an idea for proving tail properties of Markov chains. By Lemma 6.2, to prove

$$\mathbf{P}\big[\exists \delta \in \mathfrak{S}\ \exists T \in\ N\ \forall t \geq T \quad U_t \models \mathcal{Q}(\delta)\big] = 1,$$

it is sufficient to find a constant $d \in \mathbb{N}$ and a constant $\kappa \in (0, 1]$ such that

$$\mathbf{P}\big[M_{d\cdot(k+1)} = \infty \big| M_{d\cdot k} = m < \infty\big] \geq \kappa > 0 \tag{6.28}$$

for all k and m. Where M_k is the k-th t such that

$$\forall \delta \in \mathfrak{S} \quad \neg\Big(U_t \models \mathcal{Q}(\delta) \text{ and } U_{t+1} \models \mathcal{Q}(\delta)\Big).$$

For a fixed d, to show (6.28) we only need to seek for a suitable $\delta_0 \in \mathfrak{S}$ such that

$$\mathbf{P}\big[\forall t \geq m + d \quad U_t \models \mathcal{Q}(\delta_0) \ \big|\ M_{d\cdot k} = m < \infty\big] \geq \kappa > 0. \tag{6.29}$$

And due to the Markov property of the process, it is often easy for us to find a $\delta_0 \in \mathfrak{S}$ which shows (6.29). For example, in the proof of Theorem 6.1, we use best solution $\mathbf{X}_t^{ITM} \in \mathbf{X}_t \cup \mathbf{M}_t$ as the δ_0 for proving absorption of solutions in the cases NM, GTMU and LTMU with WO, and for the case LTMU with FIFO we select the best solution $\mathbf{X}_t^{IT} \in \mathbf{X}_t$ as the δ_0.

Applying this idea, we may also easily show 'absorption of solutions' for heuristic algorithms of solution-based type. We take GA as an example. As we mentioned in the beginning of this Section, absorption of solutions are named as genetic drift in GA. To study genetic drift, we assume conventionally a GA *without mutation operator* e.g. [AM94]. Let P_t be the starting population in iteration t. Obviously, $(P_t)_{t=0,1,2,\ldots}$ forms a Markov chain. Define $\mathcal{Q} = \{\mathcal{Q}(s) \mid s \in S\}$ and $P_t \models \mathcal{Q}(s)$ if and only if $P_t = (s, \ldots, s)$. Then genetic drift corresponds to tail property $\mathcal{Q}$. Recall that in each iteration of GA, $M/2$ pairs of solutions where M is the population size for children, say

$$\mathbf{S}_t := \big((s^{(1)}, s^{(2)}), \ldots, (s^{(M-1)}, s^{(M)})\big)$$

are randomly picked out from the present population P_t with a distribution based on solutions qualities. Then, each pair produces two children by a fixed crossover operator. Let O_t be the M children produced by pairs in $\mathbf{S}_t$. Here, we do not consider mutation. So the next population P_{t+1} is directly selected from $P_t \cup O_t$ either by a truncate selection (i.e. the so-called $\lambda+\mu$ rule) or a quality-based random selection (i.e. the so-called (λ, μ) rule). Similarly, due to the finiteness of S, we may assume that there exists a $\beta \in (0, 1]$ which does not depend on the contents of P_t such that for all $t \geq N$,

$$\begin{aligned}\mathbf{P}\Big[\mathbf{S}_t = \big((\mathbf{X}_t^{IT}, &\mathbf{X}_t^{IT}) \ldots, (\mathbf{X}_t^{IT}, \mathbf{X}_t^{IT})\big) \ \big|\ P_t\Big] \\ &= \mathbf{P}\big[\mathbf{S}_t = \big((\mathbf{X}_t^{IT}, \mathbf{X}_t^{IT}) \ldots, (\mathbf{X}_t^{IT}, \mathbf{X}_t^{IT})\big)\big] \geq \beta > 0,\end{aligned}$$

where $\mathbf{X}_t^{IT}$ is the best solution in P_t. Therefore, if we construct P_{t+1} by a truncate selection, then we can show that

$$\begin{aligned}&\mathbf{P}\big[\forall t \geq m+2 \quad P_t \models \mathcal{Q}(\mathbf{X}_{m+1}^{IT}) \mid M_{2\cdot k} = m < \infty\big] \\ &\quad \geq \mathbf{P}\Big[\mathbf{S}_{m+1} = \big((\mathbf{X}_{m+1}^{IT}, \mathbf{X}_{m+1}^{IT}), \ldots, (\mathbf{X}_{m+1}^{IT}, \mathbf{X}_{m+1}^{IT})\big) \;\Big|\; M_{2\cdot k} = m < \infty\Big] \geq \beta > 0,\end{aligned}$$

for any m and k. So, genetic drift holds for GA with truncate selection in population update. Although the discussion here uses a 'dirty' trick, it reflects a truth in fact. Similar discussion may apply to GA with random selection in the population update.

6.2.3 On the absorbing solution $s_{<\infty}$

In the above, we showed absorption of solutions in the framework for the case of $\rho_t \geq \rho > 0$ under different memory update rules. But we do not get any insight into the absorbing solution s_∞. Now, we are to show some properties for $s_{<\infty}$.

Theorem 6.3 below shows that conditioned on absorption of solutions, the memories are also absorbed to $s_{<\infty}$. As a result, if we use GTMU or LTMU with WO to update memory, then $s_{<\infty}$ must be the *best solution found.* And if we use NM or LTMU with FIFO to update memory, $s_{<\infty}$ must be an *iteration best solution.*

Theorem 6.3. *Assume that absorption of solutions and* (6.3) *hold, and we use NM, GTMU, LTMU with WO or LTMU with FIFO to update memory. Then absorption of memories also holds almost surely i.e.*

$$\exists T \in \mathbb{N}\ \forall t \geq T \quad \mathbf{M}_t = (s_{<\infty}, \ldots, s_{<\infty})$$

where $s_{<\infty}$ is the solution which the sampling is absorbed to.

Proof. Obviously, if we use NM or LTMU with FIFO, Theorem 6.3 holds. We only need to prove for the cases of GTMU and LTMU with WO.

We assume that the employed memory update rule is GTMU or LTMU with WO, and

$$\mathbf{P}\big[\forall t \geq T \quad \mathbf{X}_t = (s_{<\infty}, \ldots, s_{<\infty})\big] = 1 \tag{6.30}$$

for some random time $T \in \mathbb{N}$ and absorbing solution $s_{<\infty} \in S$. We want to show that

$$\mathbf{P}\big[\ \mathbf{X}_T^{ITM} = s_{<\infty}\big] = 1, \tag{6.31}$$

where $\mathbf{X}_T^{ITM}$ also denotes the best solution in $\mathbf{M}_T \cup \mathbf{X}_T$.

Note that if (6.31) holds, then by (6.30) and assumption (5.7) the Theorem holds. I.e. by (6.30) and (6.31) we have

$$\mathbf{P}\big[\forall t \geq T \quad \mathbf{X}_t = (s_{<\infty}, \ldots, s_{<\infty}),\ \mathbf{X}_t^{ITM} = s_{<\infty}\big] = 1.$$

And event

$$\big[\forall t \geq T \quad \mathbf{X}_t = (s_{<\infty}, \ldots, s_{<\infty}),\ \mathbf{X}_T^{ITM} = s_{<\infty}\big]$$

implies that there exists $T' > T$,

$$\left[\forall t \geq T' \quad \mathbf{M}_t = (s_{<\infty}, \ldots, s_{<\infty})\right]$$

under assumption (5.7), since we use GTMU or LTMU with WO to update memories.

Observe that

$$\begin{aligned} \mathbf{P}\Big[\exists t \geq T \quad \mathbf{X}_t \neq (s_{<\infty}, \ldots, s_{<\infty}) \;\Big|\; \mathbf{X}_T^{ITM} \neq s_{<\infty}\Big] & \\ \geq \mathbf{P}\Big[X_{T+1} \neq (s_{<\infty}, \ldots, s_{<\infty}) \;\Big|\; \mathbf{X}_T^{ITM} \neq s_{<\infty}\Big] & \\ \geq \mathbf{P}\Big[X_{T+1} = (\mathbf{X}_T^{ITM}, \ldots, \mathbf{X}_T^{ITM}) \;\Big|\; \mathbf{X}_T^{ITM} \neq s_{<\infty}\Big] & \\ \geq \gamma_1 \cdot \left(\lambda \cdot \alpha \cdot \rho_{T+1}\right)^{L \cdot N} > 0, & \end{aligned} \tag{6.32}$$

where we use assumption (6.3) and (5.10), and the last inequality in (6.32) follows by a completely analogous discussion as in steps (6.9)-(6.13) in the proof of Theorem 6.1.

By (6.32), we have

$$\mathbf{P}\left[\mathbf{X}_T^{ITM} \neq s_{<\infty}\right] = 0.$$

Otherwise,

$$\begin{aligned} &\mathbf{P}\Big[\exists t \geq T \quad \mathbf{X}_t \neq (s_{<\infty}, \ldots, s_{<\infty})\Big] \\ &\qquad = \mathbf{P}\Big[\exists t \geq T \quad \mathbf{X}_t \neq (s_{<\infty}, \ldots, s_{<\infty}) \;\Big|\; \mathbf{X}_T^{ITM} \neq s_{<\infty}\Big]\mathbf{P}\left[\mathbf{X}_T^{ITM} \neq s_{<\infty}\right] > 0 \end{aligned}$$

which contradicts (6.30). This completes the proof. □

By Theorem 6.3, we know that $s_{<\infty}$ must be of a high quality if we compare it only with solutions seen in the search history. But in optimization, we may be more interested in whether $s_{<\infty} \in S^*$. Or in other word, whether absorption of solutions and finite reachability are compatible. Note that finite reachability concerns whether the framework can reach an optimal solution in finitely many iterations (effectiveness), and absorption of solutions may be related to the number of possible solutions visited by the framework (efficiency). Hence, compatibility of absorption of solutions and finite reachability may reflect a theoretical possibility of perfectly balancing efficiency and effectiveness.

Obviously, the setting $\rho_t \geq \rho > 0$ conflicts the necessary condition of finite reachability i.e.

$$\sum_{t=1}^{\infty} \prod_{m=1}^{t} (1 - \rho_m) = \infty,$$

see also b) of Theorem 5.7. This means that when absorption of solutions holds, finite reachability may not be guaranteed. This may give an intuition that finite reachability and absorption of solutions are incompatible. Theorem 6.4 below confirms this theoretically, at least in the case of a unique optimal solution.

Theorem 6.4 shows that for the case of a unique optimal solution, absorption of solutions and finite reachability are not compatible, i.e. we can not find a strategy

which guarantees both absorption of solutions and finite reachability. Consequently, $\mathbf{P}[s_{<\infty} \notin S^*] > 0$ holds at least in the case of a unique optimal solution.

Theorem 6.4. *Assume $|S^*| = 1$, then absorption of solutions and finite reachability are mutually exclusive.*

Proof. Assume that $s^* = (s_1^*, \ldots, s_L^*)$ is the unique optimal solution. We use $[\mathbf{X}_t^{(\cdot)}(1) \neq s_1^*]$ to abbreviate $[\mathbf{X}_t^{(n)}(1) \neq s_1^* \text{ for all } n \in \{1, \ldots, N\}]$.

We first derive a more strict necessary condition for finite reachability. Observe that $[\mathbf{X}_t^{(\cdot)}(1) \neq s_1^*]$ for all $t \in \mathbb{N}$ implies $\tau = \infty$. Hence, $\mathbf{P}(\tau < \infty) = 1$ requires $\mathbf{P}\big(\mathbf{X}_t^{(\cdot)}(1) \neq s_1^* \text{ for all } t \geq 0\big) = 0$. Denote $\mathfrak{X}_t$ for $[X_m^{(\cdot)}(1) \neq s_1^*, m = 0, \ldots, t-1]$, then

$$\begin{aligned} 0 &= \mathbf{P}\big(\mathbf{X}_t^{(\cdot)}(1) \neq s_1^* \text{ for all } t \in \mathbb{N} \geq 0\big) \\ &= \mathbf{P}\big(\mathbf{X}_0^{(\cdot)}(1) \neq s_1^*\big) \prod_{t=1}^{\infty} \mathbf{P}\big[\mathbf{X}_t^{(\cdot)}(1) \neq s_1^* \big| \mathfrak{X}_t\big] \end{aligned} \tag{6.33}$$

Observe that using Lemma 5.5 for $t \geq 1$

$$\begin{aligned} \mathbf{P}\big[X_t^{(\cdot)}(1) \neq s_1^* \big| \mathfrak{X}_t\big] &= \mathbf{E}\Big[\big(1 - Q(s_1^*; \diamond, 1, \mathbf{\Pi}_t)\big)^N \big| \mathfrak{X}_t\Big] \\ &\geq \mathbf{E}\Big[\Big(1 - \hbar\big(Q'(s_1^*; \diamond, 1, \mathbf{\Pi}_t)\big)\Big)^N \big| \mathfrak{X}_t\Big] \\ &= \Big[1 - \hbar\big(Q'(s_1^*; \diamond, 1, \mathbf{\Pi}_0) \prod_{m=1}^{t} (1 - \rho_m^{\diamond})\big)\Big]^N, \end{aligned} \tag{6.34}$$

where we use the fact that under condition $\mathfrak{X}_t$ we have $\mathbf{W}_m(s_1^*, 1) = 0$ for $m = 0, \ldots, t-1$ by assumption (5.8) and fact that $M_0 = \emptyset$. Hence we may apply Lemma 5.6 d) with $w = 0$ and $i = 0, y = \diamond$, $T = 0$ to obtain the last equality in (6.34). Recall that $\rho_m^{\diamond} = \rho_m / \sum_{a' \in C_0(\diamond)} \mathbf{\Pi}_m(a', 1)$.

Under assumption (5.2), (5.3) and (5.4), we know $Q(s_1^*; \diamond, 1, \mathbf{\Pi}_0) \in (0, 1)$ and hence $\mathbf{P}\big(\mathbf{X}_0^{(\cdot)}(1) \neq s_1^*\big) > 0$. Now (6.33) and (6.34) show that reachability of the optimal solution requires

$$\prod_{t=1}^{\infty} \Big[1 - \hbar\big(Q'(s_1^*; \diamond, 1, \mathbf{\Pi}_0) \prod_{m=1}^{t} (1 - \rho_m^{\diamond})\big)\Big] = 0,$$

and by Lemma 5.1 a) and Lemma 5.4 c) this is equivalent to

$$\sum_{t=1}^{\infty} \ell\big(\prod_{m=1}^{t} (1 - \rho_m^{\diamond})\big) = \infty. \tag{6.35}$$

We are now going to show that absorption implies that (6.35) does not hold. If the

algorithm is almost surely absorbed in finite time, then

$$\begin{aligned} 1 &= \mathbf{P}\big(\exists a \in \mathcal{A}\ \exists k \in \mathbb{N}\ \forall m \geq k \quad X_m^{(\cdot)}(1) \equiv a\big) \\ &= \sum_{a\in\mathcal{A}} \mathbf{P}\big(\exists k \in \mathbb{N}\ \forall m \geq k \quad X_m^{(\cdot)}(1) \equiv a\big) \\ &= \sum_{a\in\mathcal{A}} \lim_{k\to\infty} \mathbf{P}\big(\forall m \geq k \quad X_m^{(\cdot)}(1) \equiv a\big) \\ &= \lim_{k\to\infty} \sum_{a\in\mathcal{A}} \mathbf{P}\big(\forall m \geq k \quad X_m^{(\cdot)}(1) \equiv a\big). \end{aligned} \tag{6.36}$$

We may use the recursion of Lemma 5.6 c) for $y = \diamond, T = 0$ and obtain from Lemma 5.1 (5.14) that for all $m \geq k$

$$Q'(a;\diamond,1,\mathbf{\Pi}_m) \leq 1 - \big(1 - Q'(a;\diamond,1,\mathbf{\Pi}_0)\big) \prod_{l=1}^{m} (1-\rho_l^{\diamond}).$$

This may be used to deduce in (6.36)

$$\begin{aligned} \mathbf{P}\big(\forall m \geq k \quad X_m^{(\cdot)}(1) \equiv a\big) &= \mathbf{P}[X_k^{(\cdot)}(1) \equiv a] \prod_{m=k+1}^{\infty} \mathbf{P}\big(X_m^{(\cdot)}(1) \equiv a \mid \mathfrak{N}_k^m\big) \qquad (6.37) \\ &\leq \prod_{m=k+1}^{\infty} \mathbf{E}\Big[\hbar\big(Q'(a;1,\diamond,\mathbf{\Pi}_m)\big)^N \big| \ \mathfrak{N}_k^m\Big] \\ &\leq \prod_{m=k+1}^{\infty} \mathbf{E}\Big[\hbar\Big(1 - \big(1 - Q'(a;1,\diamond,\mathbf{\Pi}_0)\big) \prod_{l=1}^{m} (1-\rho_l^{\diamond})\Big)^N \big| \mathfrak{N}_k^m\Big] \\ &= \prod_{m=k+1}^{\infty} \Big[1 - \ell\Big(\big(1 - Q'(a;1,\diamond,\mathbf{\Pi}_0)\big) \prod_{l=1}^{m} (1-\rho_l^{\diamond})\Big)\Big]^N, \end{aligned}$$

where $\mathfrak{N}_k^m$ denotes the event $[\mathbf{X}_t^{(\cdot)}(1) \equiv a, \forall t = k, \ldots, m-1]$. With (6.37) and (6.36), we derive that absorption requires

$$\prod_{m=1}^{\infty} \Big[1 - \ell\Big(\big(1 - Q'(a;1,\diamond,\mathbf{\Pi}_0)\big) \prod_{l=1}^{m} (1-\rho_l^{\diamond})\Big)\Big]^N > 0$$

for at least one $a \in \mathcal{A}$ which again by Lemma 5.1 a) and Lemma 5.4 c) is equivalent to

$$\sum_{m=1}^{\infty} \ell\big(\prod_{l=1}^{m} (1-\rho_l^{\diamond}) < \infty \tag{6.38}$$

contradicting (6.35). □

Note that Theorem 6.4 does not mean that once absorption of solutions holds, no optimal solution can occur. It just means that we can not find a suitable strategy which

may almost surely lead the framework efficiently narrow its search range and simultaneously guarantee to find an optimal solution. Actually, from the runtime analysis in Section 5.4, we have seen that it is possible that the framework efficiently narrows its search range and simultaneously reaches an optimal solution in finite iterations, but the probability can not be 1.

6.3 Absorption of models

Recall that the motivation of MBS algorithms is to iteratively evolve a model so as to reach a limit concentrating on optimal solutions. Hence, the following questions may be of great importance in theoretical analysis of MBS algorithms.

Question 1 Does there exist a setting which makes $(\mathbf{\Pi}_t)_{t=0,1,2,\dots}$ convergent almost surely?

Question 2 Does there exist a setting which makes $(\mathbf{\Pi}_t)_{t=0,1,2,\dots}$ convergent to a limiting model $\mathbf{\Pi}_\infty$ concentrated on some optimal solution almost surely?

Obviously, a negative answer to Question 1 also negate Question 2. So, we inspect Question 1 first.

Theorem 6.5 below leads a positive answer to Question 1. It shows that absorption of solutions implies absorption of models. Thereby, from Theorem 6.1 if we set $\rho_t \geq \rho > 0$ for some constant ρ and all $t \in \mathbb{N}$, models converge to an one-point distribution in $\mathbb{P}$, i.e. s_∞ exists. Theorem 6.5 also says that once $s_{<\infty}$ exists, s_∞ would also exist. Moreover, we see in the proof that $s_{<\infty} = s_\infty$ provided $s_{<\infty}$ exists.

Theorem 6.5. *Assume absorption of solutions and* (6.3)*, and we use NM, GTMU, LTMU with WO or LTMU with FIFO to update memories, then absorption of models hold.*

Here, we recall that NM shorts for "non-memory", GTMU shorts for "global truncate memory update", LTMU shorts for "local truncate memory update", WO shorts for "worst out" and FIFO shorts for "first in first out". In NM, we do not use memory i.e. $\mathbf{M}_t \equiv \emptyset$. In GTMU, we use the M best solutions in $\mathbf{X}_t \cup \mathbf{M}_t$ as next memory $\mathbf{M}_{t+1}$, where M indicates the memory size. In LTMU, we first remove some solutions in $\mathbf{M}_t$ by an out rule, then form $\mathbf{M}_{t+1}$ with the resulted memory by adding the same number of best solutions in $\mathbf{X}_t$. The frequently used out rules in LTMU are WO and FIFO.

Proof of Theorem 6.5. We only show the Theorem for the case NM. Other cases are similar with observation of Theorem 6.3.

We assume that we do not use memory. Absorption of solutions formally means that there are random variables $T < \infty$ and $s_{<\infty} = \big(s_{<\infty}(1), \dots, s_{<\infty}(L)\big)$ taking on values in S such that almost surely

$$\text{for all } t > T \quad \mathbf{X}_t = \big(s_{<\infty}, \dots, s_{<\infty}\big). \tag{6.39}$$

Note that $\mathcal{N}_t^b$ is a subsample of $\mathbf{X}_t$ under NM. Hence by (6.39), it is almost surely that

$$\text{for all } t > T \quad \mathcal{N}_t^b = \big(s_{<\infty}, \dots, s_{<\infty}\big).$$

By assumption (5.8) and (5.9) this implies $\mathbf{W}_t\big(s_{<\infty}(i), i\big) = 1$ and $\mathbf{W}_t\big(s_{<\infty}(i), i\big) = 0$ for all $a \neq s_{<\infty}(i)$ and all $t > T$. Note that if there are infinitely $t \in \mathbb{N}$ such that $\rho_t = 1$, then absorption of models follows immediately. Because $\mathbf{\Pi}_t(a; i) = \mathbf{W}_{t-1}(a; i)$ when $\rho_t = 1$. So, now we only consider the case $\rho_t \in (0, 1)$ for all $t \in \mathbb{N}$.

By the necessary condition (6.38) for absorption of solutions in the proof of Theorem 6.4, we have

$$\sum_{t=1}^{\infty}\prod_{m=1}^{t}(1-\rho_m^{\diamond})<\infty.$$

This implies

$$\prod_{t=1}^{\infty}(1-\rho_t^{\diamond})=0.$$

By b) of Lemma 5.6, we have

$$\rho_t^{\diamond}\in(0,1)$$

when $\rho_t\in(0,1)$. Hence by a) of Lemma 5.1, we have

$$\sum_{t=1}^{\infty}\rho_t^{\diamond}=\infty.$$

Note that by a) of Lemma 5.6,

$$\lim_{t\to\infty}\frac{\rho_t^{\diamond}}{\rho_t}\leq\lim_{t\to\infty}\sup\,G_0(\diamond;\mathbf{\Pi}_t)\leq 1.$$

Therefore, sequence $\{\rho_t\}_{t\in\mathbb{N}}$ and $\{\rho_t^{\diamond}\}_{t\in\mathbb{N}}$ have the same convergence property. As a result

$$\sum_{t=1}^{\infty}\rho_t=\infty.$$

By a) of Lemma 5.1, we have

$$\prod_{t=1}^{\infty}(1-\rho_t)=0. \tag{6.40}$$

Hence we may use the basic recursion (4.21) and Lemma 5.1 (5.14) to obtain for all $t>T$ almost surely

$$\mathbf{\Pi}_t(a;i)\leq\mathbf{\Pi}_T(a;i)\prod_{m=T+1}^{t}(1-\rho_m)\quad\text{for } a\neq s_{<\infty}(i)$$

$$\mathbf{\Pi}_t\big(s_{<\infty}(i);i\big)\geq 1-\Big(1-\mathbf{\Pi}_T\big(s_{<\infty}(i);i\big)\Big)\prod_{m=T+1}^{t}(1-\rho_m)$$

which for $t\to\infty$ converges to the probability measure concentrated on $s_{<\infty}$ by (6.40). □

Theorem 6.5 presents a positive answer to Question 1. But it can not lead to a positive answer of Question 2, since absorption of solutions and finite reachability are mutually exclusive for the case of $|S^*|=1$.

However, [Gut02] and [Mar05] have given a positive answer to Question 2 for the case of $\mathcal{M}$ = GTMU with memory size $M = 1$. In [Gut02], W. J. Gutjahr showed that if $|S^*| = 1$, $\mathcal{M}$ = GTMU, $\mathcal{S}$ = TS (truncate selection) with subsample size $N_b = 1$, $\mathcal{L}$ = WL (weighted learning), and

$$\rho_t \leq 1 - \frac{\log t}{\log(t+1)} \ \forall t \geq 1 \quad \text{and} \quad \sum_{t=1}^{\infty} \rho_t = \infty,$$

then it follows almost surely that $\left(\mathbf{\Pi}_t\right)_{t\in\mathbb{N}}$ converges to a limiting model concentrating on the unique optimal solution. In [Mar05], L. Margolin showed the same result in the case of $|S^*| = 1$, $\mathcal{M}$ = GTMU, $\mathcal{S}$ = TS with $N_b = 1$, $\mathcal{L}$ = UL (uniform learning).

Theorem 6.6 below continues [Gut02] and [Mar05]. Particularly, it also shows a positive answer for Question 2 in the case of LTMU. Note that

$$\forall t \geq 1,\ \rho_t \leq 1 - \frac{\log t}{\log(t+1)} \quad \Rightarrow \quad \sum_{t=1}^{\infty} \prod_{m=1}^{t} (1-\rho_m)^L = \infty.$$

Therefore, Theorem 6.6 also weakens the conditions in [Gut02] and [Mar05] greatly.

Theorem 6.6. *Assume $|S^*| = 1$ and we use GTMU to update memory, and TS, MID or MRS selection to select $\mathcal{N}_t^b$ from $\mathbf{X}_t \cup \mathbf{M}_t$. Let $M = 1$ be the memory size, $N_b = 1$ be the subsample size. Then if*

$$\sum_{t=1}^{\infty} \prod_{m=1}^{t} (1-\rho_m)^L = \infty \quad \textit{and} \quad \sum_{t=1}^{\infty} \rho_t = \infty$$

then absorption of models holds, and the limiting model must concentrate on the unique optimal solution only. Furthermore, if we use LTMU with WO to update memory and TS to select subsample, and set $M > 2$, $N_b = 1$, then the conclusion still holds.

Recall that in TS selection, we take the N_b best solutions in $\mathbf{M}_t \cup \mathbf{X}_t$ as the subsample $\mathcal{N}_t^b$. In MID selection, we set $\mathcal{N}_t^b = \mathbf{M}_{t+1}$. And in MRS, we form $\mathcal{N}_t^b$ by a random selection based on solutions qualities from $\mathbf{M}_{t+1}$. Note that under GTMU, when $M = N_b = 1$, the selection TS, MID and MRS become identical i.e. they always select the best so far solution. However, when $N_b < M$, the three selections are different. In Theorem 6.7, we will see that when $N_b \leq M$, Theorem 6.6 still holds. The following only shows the proof for the case GTMU. For the case LTMU with WO (worst out rule) and TS selection, the proof is completely the same with observation to the fact that when the unique optimal solution occurs, the memory will always contain it under the worst out rule.

Proof of Theorem 6.6. Let $s^* = (s_1^*, \ldots, s_L^*)$ be the unique optimal solution. Assume that the memory update rule is GTMU, subsample selection rule is TS, MID or MRS, $M = N_b = 1$. Note that by a) of Theorem 5.7, we have

$$\mathbf{P}\big[\tau < \infty\big] = 1. \tag{6.41}$$

By (6.41), we have that

$$\mathbf{P}\big[s^* \in \mathbf{X}_T\big] = 1 \tag{6.42}$$

for some random time $T \in \mathbb{N}$. We are to show that $[s^* \in \mathbf{X}_T]$ implies $\mathbf{\Pi}_t \to \mathbf{\Pi}_\infty$ as $t \to \infty$, where

$$\mathbf{\Pi}_\infty(a;i) := \begin{cases} 1 & \text{if } a = s_i^*, \\ 0 & \text{otherwise,} \end{cases} \qquad \text{for all } a \in \mathcal{A} \text{ and } i = 1, \ldots, L.$$

Obviously, this proves the Theorem.

Recall that we use GTMU (global truncate memory update) to update memory, and memory size $M = 1$. Therefore, conditioned on $[s^* \in \mathbf{X}_T]$ we have

$$\forall t \geq T+1 \ \mathbf{M}_t = (s^*) \quad \text{ and } \quad \forall t \geq T \ \mathcal{N}_t^b = (s^*) \tag{6.43}$$

under TS (truncate selection), MID (memory identity selection) and MRS (memory random selection). By (6.43), (5.8) and (5.9), we have

$$\mathbf{W}_t(a;i) = \begin{cases} 1 & \text{if } a = s_i^*, \\ 0 & \text{otherwise,} \end{cases} \qquad \text{for all } a \in \mathcal{A},\ t \geq T \text{ and } i = 1, \ldots, L. \tag{6.44}$$

It follows immediately by (6.44) and (5.15) of Lemma 5.1 that

$$\mathbf{\Pi}_t(a;i) = \mathbf{\Pi}_T(a;i) \prod_{m=T+1}^{t} (1-\rho_m) \quad \text{if } a \neq s_i^*$$

$$\mathbf{\Pi}_t(a;i) = 1 - \big(1 - \mathbf{\Pi}_T(a;i)\big) \prod_{m=T+1}^{t} (1-\rho_m)$$

for all $s \in \mathcal{A}$, $i = 1, \ldots, L$ and $t \geq T+1$. Thereby, by Lemma a) of Lemma 5.1 and

$$\sum_{t=1}^{\infty} \rho_t = \infty,$$

we complete the proof. □

Theorem 6.6 shows an example for $\mathbf{P}[s_\infty \in S^*] = 1$. Hence, it may follow immediately from Theorems 6.4 and 6.6 that absorption of models does not necessarily mean absorption of solutions. Moreover, existence of s_∞ does not imply existence of $s_{<\infty}$, although existence of $s_{<\infty}$ implies $s_\infty = s_{<\infty}$.

Actually, we can further extend Theorem 6.6 to the case that $N_b \leq M$ i.e. subsample size is not bigger than the memory size. Note that

$$\sum_{t=1}^{\infty} \prod_{m=1}^{t} (1-\rho_m)^L = \infty \Rightarrow \rho_k \in (0,1) \text{ and } \sum_{t=t'}^{\infty} \prod_{m=t'}^{t} (1-\rho_m)^L = \infty \quad \text{for all } k, t' \geq 1. \tag{6.45}$$

By the general assumption (5.2) on $\mathbf{\Pi}_0$ and Lemma 5.2 d), we know that

$$\mathbf{\Pi}_{t'}(a;i) > 0 \quad \text{for all } t' \geq 1, a \in \mathcal{A} \text{ and } i = 1, \ldots, L \tag{6.46}$$

since each $\rho_k \in (0,1)$. Recall that the underlying stochastic process is Markov chain. By (6.46) and second part of (6.45) i.e.

$$\sum_{t=t'}^{\infty} \prod_{m=t'}^{t} (1-\rho_m)^L = \infty \quad \text{for all } t' \geq 1$$

the unique optimal solution can be visited infinite times, since the sufficient condition (Theorem 5.7 a)) for finite reachability still holds after any iteration $t' \in \mathbb{N}$. When we use GTMU to update memory, there must exist an iteration T after which the memory is filled only by copies of the unique optimal solution, due to the finite size of memory and infinite visiting times of the unique optimal solution. Therefore, under TS, MID or MRS selection, after finite iterations, the subsample is also filled only by copies of the unique optimal solution. This results in the following Theorem.

Theorem 6.7. *Assume $|S^*| = 1$. We use GTMU to update memory, and TS, MID or MRS selection to select $\mathcal{N}_t^b$ from $\mathbf{X}_t \cup \mathbf{M}_t$. Let $N_b \leq M$. If*

$$\sum_{t=1}^{\infty} \prod_{m=1}^{t} (1-\rho_m)^L = \infty \quad \textit{and} \quad \sum_{t=1}^{\infty} \rho_t = \infty$$

then absorption of models holds, and the limiting model must concentrate on some optimal solution only.

Proof. By the above discussion, we may assume a random time $T \in \mathbb{N}$ such that

$$\forall t \geq T+1 \ \mathbf{M}_t = (\underbrace{s^*, \ldots, s^*}_{M}) \quad \text{and} \quad \forall t \geq T+1 \ \mathcal{N}_t^b = (\underbrace{s^*, \ldots, s^*}_{N_b})$$

where s^* is again the unique optimal solution, we use GTMU to update memory. Then the Theorem follows in a completely same way as in the proof of Theorem 6.6. □

For the non-memory case, Question 2 is still a theoretically open question, see [CJK07]. Note that a setting for Question 2 must make finite reachability also holds. By Theorem 6.4, finite reachability and absorption of solutions are mutually exclusive. And by Theorem 6.5, absorption of solutions implies absorption of models. Hence in non-memory, a first step to answer Question 2 is either to show non-equivalence of absorption of models and absorption of solutions or to check the compatibility of finitely reachability and absorption of models.

The following is a Theorem presented in [WK14b] which shows that absorption of models and finite reachability are also compatible under non-memory case. The proof of which is very complex and tedious, we will not present it here. Please see [WK14b] for a reference.

Theorem 6.8. *Assume $\mathcal{M}$ = NM (non-memory), $\mathcal{S}$ = TS (truncate selection), $\mathcal{L}$ = UL (uniform learning). Assume that $\rho \in (0,1)$ and $c_k \in \mathbb{N}$ for $k = 0, 1, \ldots,$ with $c_0 = 1$, are chosen such that*

$$\sum_{k=1}^{\infty} c_k (1-\rho)^{kL} = \infty.$$

Define $e_k := \sum_{i=1}^{k} c_{i-1},\ k \geq 1$, and let $x_k \in (0,1)$ be any sequence such that $\sum_{k=1}^{\infty} x_k < \infty$. We may now define a learning rates sequence

$$\rho_t := \begin{cases} \rho & \text{if } t = e_k \text{ for some } k \geq 1 \\ 1 - (1-x_k)^{\frac{1}{c_k - 1}} & \text{if } e_k < t < e_{k+1} \text{ for some } k \geq 1 \end{cases} \tag{6.47}$$

for $t \geq 1$. Then absorption of models holds, it has $\mathbf{P}(\tau < \infty) = 1$, and its samples do not converge.

As an example for the values in Theorem 6.8, take an arbitrary $\rho > 0$, then one may choose $c_k = (1-\rho)^{-kL}$ to obtain

$$\sum_{k=1}^{\infty} c_k (1-\rho)^{kL} = \sum_{k=1}^{\infty} 1 = \infty.$$

For $\rho < 1/2$, one could, for example, use $c_k := 2^{kL}$.

As an end, we mention a completely different study about convergence of models in the literature. In [ZM04], Zhang et al showed that the models of a particular UMDA (our framework with $\rho_t \equiv 1$ and $\mathcal{M} = \mathrm{NM}$) almost surely converges to a limit which concentrates on optimal solutions, if the sample size N and subsample size N_b are assumed to be infinite.

Note that if we do not use memory and $\rho_t \equiv 1$, the next model $\mathbf{\Pi}_{t+1}$ is exactly the empirical distribution $\mathbf{W}_t$ learned from a subsample $\mathcal{N}_t^b$ of present solutions $\mathbf{X}_t$. If the sample size $N = \infty$, we can identify present sample $\mathbf{X}_t$ by its theoretical model $\mathbf{\Pi}_t$. And the subsample $\mathcal{N}_t^b$ is then produced by a distribution $\mathbf{\Pi}_t^{se}$ which can be defined as

$$\mathbf{\Pi}_t^{se}(s) = \frac{g(f(s))\mathbf{\Pi}_t(s)}{\sum_{y \in S} g(f(y))\mathbf{\Pi}_t(y)} \quad \text{for all } s \in S,$$

where g is a non-negative decreasing function, f is the objective function, and we consider a minimizing instance here. For example, in a truncate selection, g is defined as

$$g(x) = \begin{cases} 1 & \text{if } x < f_{\alpha, \mathbf{\Pi}_t}, \\ 0 & \text{otherwise,} \end{cases}$$

where $f_{\alpha, \mathbf{\Pi}_t}$ is a constant determined by equation

$$\mathbf{\Pi}_t\big[f(s) < f_{\alpha, \mathbf{\Pi}_t}\big] = \alpha$$

for some fixed selection rate $\alpha \in (0,1)$, and then

$$\mathbf{\Pi}_t^{se}(s) = \frac{\mathbb{1}_{\{y \in S | f(y) < f_{\alpha,\mathbf{\Pi}_t}\}}(s)\mathbf{\Pi}_t(s)}{\sum_{y \in S, f(y) < f_{\alpha,\mathbf{\Pi}_t}} \mathbf{\Pi}_t(y)} \quad \text{for all } s \in S; \tag{6.48}$$

in a random selection, $g(x)$ is typically $1/x$, and then

$$\mathbf{\Pi}_t^{se}(s) = \frac{\frac{1}{f(s)}\mathbf{\Pi}_t(s)}{\sum_{y \in S} \frac{1}{f(y)}\mathbf{\Pi}_t(y)} \quad \text{for all } s \in S. \tag{6.49}$$

With assumption that $N_b = \infty$, $\mathbf{W}_t = \mathbf{\Pi}_t^{se}$ holds almost surely, since $\mathbf{W}_t$ is the empirical distribution of $\mathcal{N}_t^b$ and $\mathbf{\Pi}_t^{se}$ is the theoretical distribution of $\mathcal{N}_t^b$. However, $\mathbf{\Pi}_{t+1} = \mathbf{W}_t$ holds when $\rho_t \equiv 1$. Therefore,

$$\mathbf{\Pi}_{t+1}(s) = \mathbf{\Pi}_t^{se}(s) = \frac{g(f(s))\mathbf{\Pi}_t(s)}{\sum_{y \in S} g(f(y))\mathbf{\Pi}_t(y)} \quad \text{for all } s \in S. \tag{6.50}$$

This means that the models process $\left(\mathbf{\Pi}_t\right)_{t=0,1,2,\ldots}$ can be explained by a deterministic recursion (6.50) if we assume $N = N_b = \infty$. By (6.48) or (6.49) instead of $\mathbf{\Pi}_t^{se}$ in (6.50), we may easily show that the models process eventually converges to a limit concentrating on optimal solutions by recursive calculations. Actually, $f_{\alpha,\mathbf{\Pi}_t}$ approaches the optimal objective value f^* as $t \to \infty$ for any $\alpha \in (0,1)$ in the case of truncate selection. And by (6.49),

$$\mathbf{\Pi}_{t+1}(s) = \frac{\frac{1}{(f(s))^t}\mathbf{\Pi}_0(s)}{\sum_{y \in S} \frac{1}{(f(y))^t}\mathbf{\Pi}_0(y)} \to \begin{cases} 0 & \text{if } s \notin S^*, \\ \frac{\mathbf{\Pi}_0(s)}{\sum_{y \in S^*} \mathbf{\Pi}_0(y)} & \text{if } s \in S^*, \end{cases} \quad \text{as } t \to \infty,$$

in the random selection case, here we assume without loss in generality that each objective value $f(s)$ is positive.

7 An experimental study

In Chapters 5, we have presented conditions for guaranteeing to reach an optimal solution (finite reachability, see Theorem 5.7), and conditions for efficiently reaching an optimal solution with an overwhelming probability (runtime results, see Theorems 5.8-5.9). In particular, we see that the popular setting $\rho_t \equiv \rho$ can not guarantee to find an optimal solution, however we can make the probability for the occurrence of optimal solutions as large as possible if we reduce the constant ρ, see Theorem 5.7 c). Moreover, we see that in the case of a constant learning rate, if we adapt the sample size to the problem size or employ restricted models, the probability for optimal solutions occurring in a polynomial runtime may approximate 1 as problem size approaches infinity, see Theorems 5.8-5.9.

In Chapter 6, we inspected the absorption behavior of the underlying process. We found that when $\rho_t > \rho > 0$, the sampling will be frozen in a single solution ($s_{<\infty}$) after finite iterations (absorption of solutions), see Theorem 6.1. By Theorem 6.3, we seen that $s_{<\infty}$ is often a high quality solution compared to those seen in history. Moreover, we seen further that absorption of solutions and finite reachability conflicts with each other at least in the case of a unique optimal solution, see Theorem 6.4. This may explain why we can not guarantee to reach an optimal solution if we use a constant learning rate. In the study of the model process, we found that absorption of solutions makes the models converge to a limit which concentrating on a single solution s_∞ (absorption of models, see Theorem 6.5), and in this case $s_\infty = s_{<\infty}$. Moreover, we found conditions under which the models converge to a limit concentrating on an optimal solution ($s_\infty \in S^*$, see Theorems 6.6-6.7), and absorption of solutions does not hold. This shows that absorption of models does not imply absorption of solutions. However, presently we can only show that absorption of models and finite reachability are compatible in the case of non-memory update, see Theorem 6.8.

This Chapter aims to verify the theoretical findings in Chapter 5 and Chapter 6. We will implement the framework on a TSP instance, and check its practical performance. However, we will concentrate only on the case of a constant learning rate i.e. $\rho_t \equiv \rho > 0$. Although we find some good properties for the case of non-constant learning rates, see Theorem 5.7 a)-b) and Theorems 6.6-6.7, we are presently not able to show these effects in our practical experiments. However, our experimental results will clearly demonstrate Theorem 5.7 c) i.e.

$$\rho_t \equiv \rho > 0 \Rightarrow \mathbf{P}\big[\tau < \infty\big] < 1, \quad \text{but } \mathbf{P}\big[\tau < \infty\big] \to 1 \text{ as } \rho \to 1$$

where recall that τ is the first hitting time for optimal solutions. Also, we are not able to experimentally show the polynomially runtime results see Theorems 5.8-5.9, since they may hold only for an extremely large problem size.

The experimental results also witness absorption of solutions and models. We will see that the magnitude of the constant learning rate ρ heavily affects the absorbing time T and the absorbing solution $s_{<\infty}$, where recall that $s_{<\infty}$ coincides with s_∞ when absorption of solutions holds (see Theorem 6.5). Moreover, we shall see that when we introduce some *greedy insight* into the feasibility construction, the occurring frequency of optimal solutions will be significantly increased, simultaneously the absorbing time will be decreased.

To simplify our discussion, we will fix the memory update rule as NM i.e. $\mathbf{M}_t \equiv \emptyset$, the subsample selection rule as TS i.e. truncate selection, and the learning rule as UL i.e. uniform learning. That means that in the experiment study, we will learn the empirical model $\mathbf{W}_t$ by the relative frequencies in some elite solutions $\mathcal{N}_t^b$ choosing from present sample $\mathbf{X}_t$.

7.1 Random tour generation and learning

As mentioned, we will verify the findings on a TSP instance. We now explain the random solution generation in the experimental study, and the learning of empirical models for this particular instance.

Assume that the instance has n cities, therefore has possibly n^2 arcs. We label the cities as $0, 1, 2, \ldots, n-1$. Then, the arcs are correspondingly labeled by pairs

$$(0,0),(0,1),\ldots,(0,n-1);\cdots;(n-1,0),(n-1,1),\ldots,(n-1,n-1).$$

Here, a feasible solution is a tour which starts from a city, traverses each of other cities exactly once and then returns back to the start city. Formally, a feasible tour is a *sequence of arcs*

$$(i_0,i_1),(i_1,i_2),\ldots,(i_{n-2},i_{n-1}),(i_{n-1},i_0) \tag{7.1}$$

where $(i_0, i_1, \ldots, i_{n-2}, i_{n-1})$ is a *permutation* of $\{0, 1, 2, \ldots, n-1\}$. Obviously, a permutation of $\{0, 1, 2, \ldots, n-1\}$ also uniquely determines a feasible tour.

In the experimental study, we represent each feasible tour as a permutation of the cities (string over cities). For a feasible tour $s = (i_0, i_1, \ldots, i_{n-2}, i_{n-1})$, the traveling cost is calculated as

$$f(s) = \sum_{l=0}^{n-2} d(i_l, i_{l+1}) + d(i_{n-1}, i_0)$$

where each $d_{i_l,i_{l+1}}$ is the distance between cities i_l and i_{l+1}. We will select a TSP instance consisting of 38 cities in a country. The distance is then calculated according to their geographical locations.

7.1.1 Random solution generation

In the experimental study, the model $\mathbf{\Pi}_t$ for each iteration $t \in \mathbb{N}$ is a matrix

$$\big(\pi_t(i,j)\big)_{i=0,1,2\ldots,n-1,j=0,1,2,\ldots,n-1},$$

such that $\sum_{j=0}^{n-1} \pi_t(i,j) = 1$ for all $i = 0, 1, 2, \ldots, n-1$. We construct a random solution by iteratively extending a *partial* permutation.

In the case of a *non-greedy* feasibility construction, for a partial permutation $y = (i_0, i_1, \ldots, i_l)$ with $l < n-1$, we select a feasible continuation $i_{l+1} \in C_l(y) = \{0, 1, 2, \ldots, n-1\} - \{i_0, \ldots, i_l\}$ with a distribution

$$Q\big(a; y, l+1, \mathbf{\Pi}_t\big) = \begin{cases} \frac{\pi_t(i_l,a)}{\sum_{b \in C_i(y)} \pi_t(i_l,b)} & \text{if } a \in C_i(y), \\ 0 & \text{otherwise,} \end{cases} \tag{7.2}$$

for all $a = 0, 1, 2, \ldots, n-1$. Here, if $y = \diamond$, we randomly choose a starting city, i.e. we start with an arbitrary city.

In the case of *greedy* feasibility construction, we define feasibility distributions according to the distance information as,

- $C_0(\diamond; a) = 1/n$ for each city a, i.e. we start with an arbitrary city;
- $\sum_{a=0}^{n-1} C_i(y; a) = 1$, and

$$C_i(y; a) = \begin{cases} \frac{\big[1/d(i_l,a)\big]^\beta}{\sum_{b \in C_i(y)} \big[1/d(i_l,b)\big]^\beta} & \text{if } a \in C_i(y), \\ 0 & \text{otherwise,} \end{cases}$$

  for each partial permutation $y = (i_0, i_1, \ldots, i_l)$ with $l < n-1$, where $C_l(y) = \{0, 1, 2, \ldots, n-1\} - \{i_0, \ldots, i_l\}$.

Then, the selection probability for a feasible continuation $i_{l+1} \in C_i(y)$ to a partial permutation $y = (i_0, i_1, \ldots, i_l)$ with $l < n-1$ becomes

$$\frac{\pi_t(i_l, i_{l+1})\big[1/d(i_l, i_{l+1})\big]^\beta}{\sum_{b \in C_i(y)} \pi_t(i_l, b)\big[1/d(i_l, b)\big]^\beta}. \tag{7.3}$$

Obviously, as $\beta \to \infty$, the probability (7.3) approximates 0 if $d(i_l, i_{l+1}) \neq \min\{d(i_l, b) \mid b \in C_i(y)\}$. And when $\beta = 0$, the feasible continuation is chosen only by the present model $\mathbf{\Pi}_t$. Therefore, when β is bigger, the feasible continuation may be more likely the nearest continuation i.e. the greedy information dominates the choice of the continuation in the feasibility construction. When β is small, the present model may dominate the choice of the continuation. In the literature of ACO, see [DMC96], [DG97a], [DG97b] etc, β is generally taken to be 5.0. However, we find that $\beta = 6.0$ is better for our test instance. So, in the sequel of this Chapter, we will fix the constant $\beta = 6.0$.

Note that, as discussed in Subsection 4.2.3, these two samplings can be both stated in a more standard terms of feasibility construction, i.e. we can restate them by representing the tours as strings over arcs.

7.1.2 Learning the empirical distribution $\mathbf{W}_t$

In the experimental study, we use uniform learning, and the subsample $\mathcal{N}_t^b$ is truncate selected from the present sample $\mathbf{X}_t$. Let $\mathbf{X}_t^{(1)}, \ldots, \mathbf{X}_t^{(N)}$ be the present solutions.

According to their traveling costs, we order them as

$$f(\mathbf{X}_t^{(l_1)}) \leq \ldots \leq f(\mathbf{X}_t^{(l_N)})$$

where f is the traveling cost function. We collect the N_b best solutions in $\mathcal{N}_t^b$ i.e.

$$\mathcal{N}_t^b = \big(\mathbf{X}_t^{(l_1)}, \ldots, \mathbf{X}_t^{(l_{N_b})}\big).$$

Here, we write each solution as permutation i.e.

$$\mathbf{X}_t^{(l_m)} = (i_{t,0}^{l_m}, i_{t,1}^{l_m}, \ldots, i_{t,n-1}^{l_m})$$

is a permutation of $0, 1, 2, \ldots, n-1$, for $m = 1, \ldots, N_b$. By uniform learning, $\mathbf{W}_t = \big(\mathbf{W}_t(i,j)\big)_{i=0,1,2,\ldots,n-1;j=0,1,2,\ldots,n-1}$ is calculated as

$$\mathbf{W}_t(i,j) = \frac{\sum_{s \in \mathcal{N}_t^b} \mathbb{1}_{\{(a,b)\in E \mid (a,b)\in s\}}(i,j)}{N_b}$$

where E is collection of arcs, $(a,b) \in s$ means that arc (a,b) is on tour $s = (i_0, i_1, \ldots, i_{n-1})$ i.e.

$$\exists j = 0, 1, \ldots, n-2 \quad a = i_j, b - i_{j+1} \quad \text{or } a = i_{n-1}, b = i_0.$$

Obviously, $\mathbf{W}_t(i,j)$ is the relative frequency of arc (i,j) in $\mathcal{N}_t^b$. After learning, the next model $\mathbf{\Pi}_{t+1} = \big(\pi_{t+1}(a,b)\big)_{(a,b)\in E}$ is constructed as

$$\pi_{t+1}(a,b) = (1-\rho)\pi_t(a,b) + \rho \mathbf{W}_t(a,b)$$

for each $(a,b) \in E$, where $\rho \in (0,1]$ is constant learning rate. In the experimental, we want to see how ρ affects the performance resp., in the two feasibility constructions (7.2) and (7.3).

7.2 Experimental study

7.2.1 The test instance

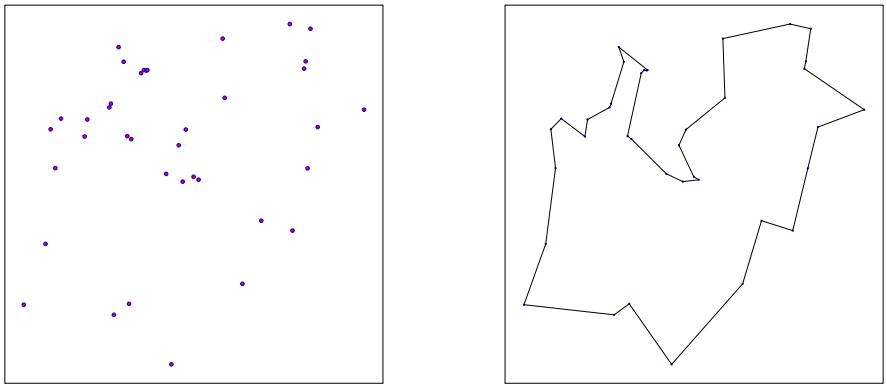
The test instance is a TSP problem of 38 cities i.e. $n = 38$, called $dj38$ in the world TSP library:

$$http://www.math.uwaterloo.ca/tsp/world/.$$

The Figure 7.1 (a)-(b) shows resp., its corresponding points set and an optimal tour. The optimal tour is of length 6659.43.[1] The geographical locations of the 38 cities are listed in Table 7.1. The distances between different solutions are calculated according to their geographical locations. For example, the distance between city 0 and 1 is

$$\begin{aligned} d(0,1) &= \sqrt{(x_0 - x_1)^2 + (y_0 - y_1)^2} \\ &= \sqrt{(11003.611100 - 11108.611100)^2 + (42102.500000 - 42373.888900)^2} \\ &\approx 290.993. \end{aligned}$$

[1] The value for this optimal tour reported in the library is 6656, however the value under C++ 64 bit is 6659.43.



(a) points set (b) An optimal tour

Figure 7.1: A TSP test instance

The size of the feasible set for this instance (i.e. total number of possible feasible tours) is

$$37!/2 = 6881876545613172523157989790790451200000000 \approx 6.88188 \times 10^{42}.$$

7.2.2 Experimental setting

We take sample size $N = 100$ and subsample size $N_b = 5$. We do not use memory, use truncate selection and the uniform learning. The first model

$$\mathbf{\Pi}_0 = \big(\pi_0(i,j)\big)_{i=0,1,2,\ldots,37;j=0,1,2,\ldots,37}$$

is taken to be 'uniform' i.e. each $\pi_0(i,j) = 1/38 \approx 0.026315$. The framework is written in C++ with Qt, and implemented on a 64 bit machine under Linux system. We test the framework under the two different random solution generations i.e. non-greedy feasibility construction (7.2) and greedy feasibility construction (7.3). For each case, we test the framework with several different constant learning rates.

7.2.3 Results under non-greedy feasibility construction

Under the non-greedy feasibility construction (7.2), we implement the framework on $dj38$ under 21 different constant learning rates see (Table 7.2). For each constant learning rate, we do 50 independent trials, i.e. we did totally 21×50 trials under non-greedy feasibility construction. For each trial, we *stop* the framework if the sampling has been frozen at a fixed solution for 1000 consecutive iterations. Figure Table 7.2 collects the experimental results.

Each row of Table 7.2 reports the average performance over the 50 independent trials for a corresponding constant learning rate. The first column lists all the rates we employed. The column $f(s^*)$ lists the optimal cost value as a reference. Column $Fre(s^*)$

cities	x_i (x-coordinate)	y_i (y-coordinate)
0	11003.611100	42102.500000
1	11108.611100	42373.888900
2	11133.333300	42885.833300
3	11155.833300	42712.500000
4	11183.333300	42933.333300
5	11297.500000	42853.333300
6	11310.277800	42929.444400
7	11416.666700	42983.333300
8	11423.888900	43000.277800
9	11438.333300	42057.222200
10	11461.111100	43252.777800
11	11485.555600	43187.222200
12	11503.055600	42855.277800
13	11511.388900	42106.388900
14	11522.222200	42841.944400
15	11569.444400	43136.666700
16	11583.333300	43150.000000
17	11595.000000	43148.055600
18	11600.000000	43150.000000
19	11690.555600	42686.666700
20	11715.833300	41836.111100
21	11751.111100	42814.444400
22	11770.277800	42651.944400
23	11785.277800	42884.444400
24	11822.777800	42673.611100
25	11846.944400	42660.555600
26	11963.055600	43290.555600
27	11973.055600	43026.111100
28	12058.333300	42195.555600
29	12149.444400	42477.500000
30	12286.944400	43355.555600
31	12300.000000	42433.333300
32	12355.833300	43156.388900
33	12363.333300	43189.166700
34	12372.777800	42711.388900
35	12386.666700	43334.722200
36	12421.666700	42895.555600
37	12645.000000	42973.333300

Table 7.1: Point sets for test instance

$\rho_t \equiv \rho$	$f(s^*)$	$Fre(s^*)$	$\bar{\epsilon}$	$\text{Ave}_{f(\mathbf{X}^*)}$	$\text{Ave}_{T_{ab}}$	$\text{Ave}_{f(s_{<\infty})}$	Diff
1.00	6659.43	0	0.394479	9286.43	21.56	9298.4	11.97
0.95	6659.43	0	0.399518	9319.99	27.66	9322.24	2.64
0.90	6659.43	0	0.371696	9134.71	30.94	9136.43	1.2
0.85	6659.43	0	0.331936	8869.94	34.64	8878.09	8.15
0.80	6659.43	0	0.328991	8850.32	38.22	8852.39	2.06
0.75	6659.43	0	0.320315	8792.54	41.48	8795.69	3.15
0.70	6659.43	0	0.290637	8594.9	44.22	8599.37	4.47
0.65	6659.43	0	0.2667	8435.5	49.2	8438.27	2.77
0.60	6659.43	0	0.247458	8307.36	53.9	8313.86	6.5
0.55	6659.43	0	0.23021	8192.5	61.98	8195.39	2.89
0.50	6659.43	0	0.202434	8007.53	69.64	8014.62	7.09
0.45	6659.43	0	0.173463	7814.59	79.4	7820.45	5.86
0.40	6659.43	0	0.149557	7655.39	95.74	7656.44	1.05
0.35	6659.43	0	0.118443	7448.19	110.5	7453.03	4.84
0.30	6659.43	0	0.0929187	7278.22	131.96	7279.19	0.97
0.25	6659.43	0	0.059581	7056.21	167.56	7060.09	9.88
0.20	6659.43	0	0.0486467	6983.39	215.06	6988.92	5.53
0.15	6659.43	4	0.0303003	6861.21	286.64	6863.44	2.23
0.10	6659.43	13	0.0124708	6742.48	453.24	6742.52	0.04
0.05	6659.43	27	0.0067184	6704.17	930.44	6704.19	0.02
0.01	6659.43	45	0.000791341	6664.7	5051.62	6664.7	0.00

Table 7.2: Implementation results for non greedy case

records the number of trials (among the 50 independent trials) in which we observed an optimal solution, for each ρ. For example, when $\rho = 0.10$, we observed optimal solutions in 13 trials. Due to the independence of the 50 trials, $Fre(s^*)/50$ is an unbiased estimator for $\mathbf{P}[\tau < \infty]$. Column $\text{Ave}_{f(\mathbf{X}^*)}$ is the average cost value of the 50 best solutions found resp., in the 50 independent trials. Column $\bar{\epsilon}$ is the relative error of $\text{Ave}_{f(\mathbf{X}^*)}$ i.e.

$$\bar{\epsilon} = \frac{\text{Ave}_{f(\mathbf{X}^*)} - f(s^*)}{f(s^*)}.$$

Column $\text{Ave}_{T_{ab}}$ reports the average of the 50 absorbing times observed resp., in the 50 independent trials. Here, we define the absorbing time for one trial as the first iteration, from which the sampling always produces a fixed solution $s_{<\infty}$. Column $\text{Ave}_{f(s_{<\infty})}$ reports the average cost value of the 50 absorbing solutions $s_{<\infty}$ observed resp., in the 50 trials. Column Diff is the difference of $\text{Ave}_{f(s_{<\infty})}$ and $\text{Ave}_{f(\mathbf{X}^*)}$ i.e. Diff=$\text{Ave}_{f(s_{<\infty})}$-$\text{Ave}_{f(\mathbf{X}^*)}$.

Witness for absorption of solutions and models

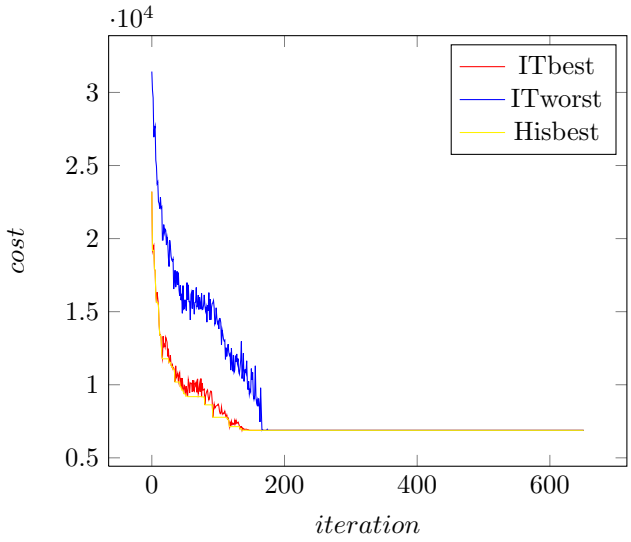


Figure 7.2: Plots of the iteration best, iteration worst and best so far costs

To check the possible absorption behavior, we arbitrarily take out 1 trial from the 50 trials under $\rho = 0.3$. And we draw three plots in Figure 7.2 resp., for the costs of iteration worst solution (blue line), iteration best solution (red line) and best solution so far (yellow line) observed in this trial. Recall that, in each trial, the sample size $N = 100$ i.e. we sample 100 random solutions. For each iteration, the *iteration worst* solution is the worst solution among the 100 solutions sampled in present iteration, the *iteration best* solution is the best solution among these 100 solutions, and the *best solution so far*

represents the best solution among all the solutions sampled in previous iterations as well as the present iteration. Obviously, these three solutions may vary as iterations. The three plots in Figure 7.2 shows resp., the variations in their costs as iterations. The x-axis represents the iteration, and y-axis represents the cost value. The three solutions have different costs in beginning. And all the three costs decrease as time goes. The three lines then coincide in an iteration near iteration 200 i.e. the costs become identical from then on. Hence, the 100 solutions sampled in iteration $t \geq 200$ are always of a fixed cost. This gives an initial impression on absorption of solutions. However, this is not sufficient to be a witness for absorption of solutions, since different solutions may have a same cost value. To clearly demonstrate the absorptions, we also take out six iterations, i.e. iterations $0, 50, 100, 150, 200, 300$, in this trial. For each of the selected 6 iterations, we draw two pictures, one for the 100 tours (solutions) sampled in that iteration and the other one for the underlying models producing these solutions. We collect the resulted 12 pictures in Figures 7.3-7.4.

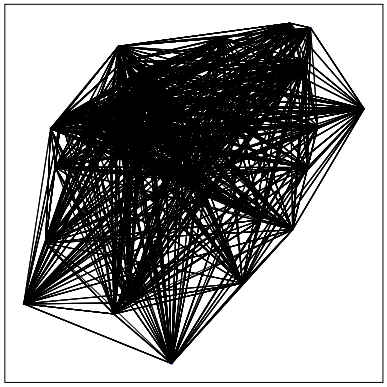
For each iteration, we draw the 100 tours according to the points set in Table 7.1 together in one picture, and we draw the corresponding model as a plot in another picture. Recall that, the model for iteration t is

$$\mathbf{\Pi}_t = \big(\pi_t(i,j)\big)_{i,j=0,1,2,\ldots,37}.$$

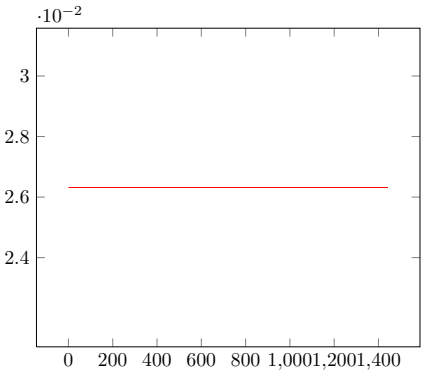
In the plot, we further encode each arc (i, j) as an integer $38 \times i + j$, and use this integer as the x-axis and the probability $\pi_t(i, j)$ as y-axis. If absorption of solutions holds, then the 100 tours should finally become identical, and show only one tour in the picture. And if absorption of models holds, the models should finally become a binary string, and show several vertical lines of length 1 in the picture.

Initially, we take a uniform model i.e. $\pi_0(i, j) \equiv 1/38$, therefore the plot of the model is a horizontal line, see Figure 7.3 (b). As a result, the 100 tours for iteration 0 is purely randomly generated, therefore they show a rather disorganized appearance in Figure 7.3 (a). When $t = 50$, the model shows a clear tendency to be a binary string (see Figure 7.3 (d)), and many arcs on the 100 sampled solutions coincide (see 7.3 (c)). When $t = 100$, the model show a more clear tendency to be a binary string (see 7.3 (f)), and more arcs on 100 solutions sampled in this iteration coincide (see 7.3 (e)).

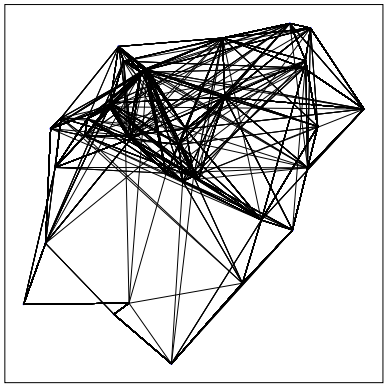
When $t = 150$, the model almost becomes a binary string (see 7.4 (b)), and most arcs on 100 solutions in this iteration coincide except for few arcs in the left top corner (see 7.4 (a)). When $t = 200$, absorption of solutions has already occurred i.e. the 100 solutions become identical (see 7.4 (c)), the model approximates a binary string (see 7.4 (d)). The pictures for iteration 300 are the same as the pictures for iteration 200, see Figure 7.4 (e)-(f). This further confirms the absorption of solutions and models.



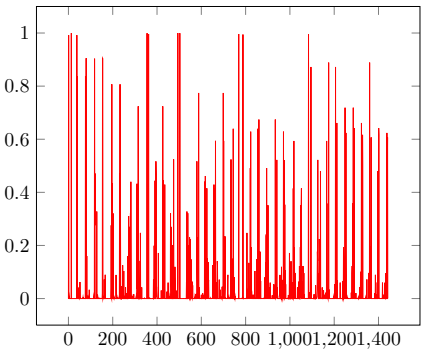
(a) Iteration 0: 100 tours



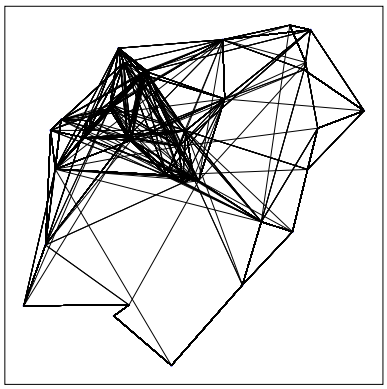
(b) Iteration 0: model



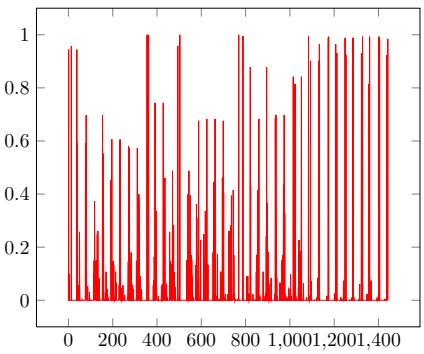
(c) Iteration 50: 100 tours



(d) Iteration 50: model

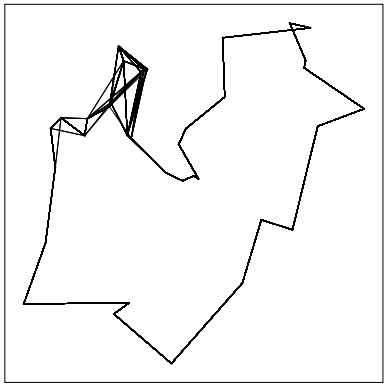


(e) Iteration 100: 100 tours

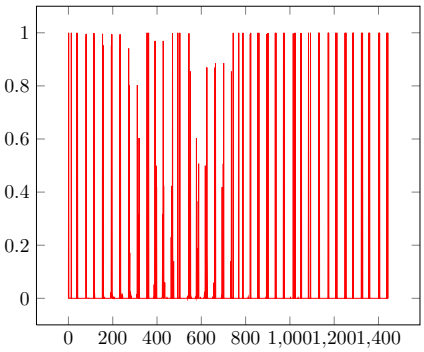


(f) Iteration 100: model

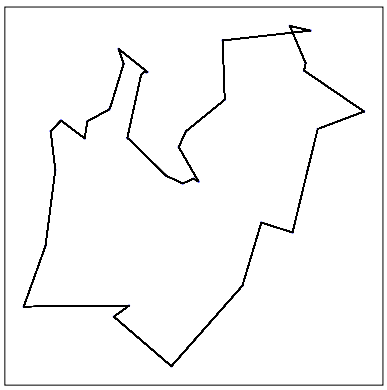
Figure 7.3: Observation of absorption of solutions and models



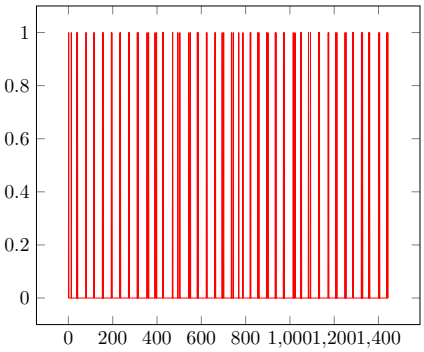
(a) Iteration 150: 100 tours



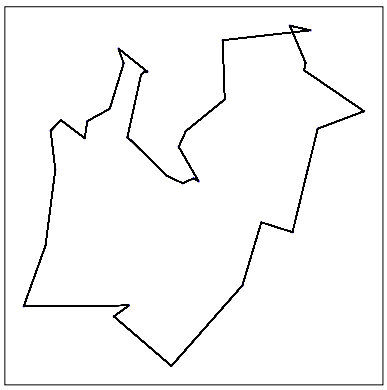
(b) Iteration 150: model



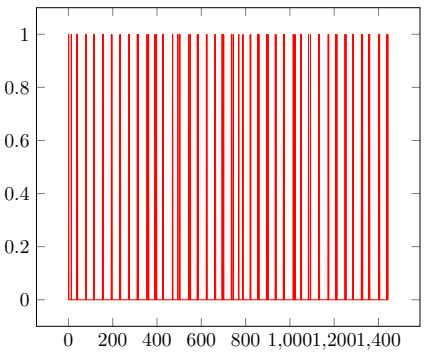
(c) Iteration 200: 100 tours



(d) Iteration 200: model



(e) Iteration 300: 100 tours

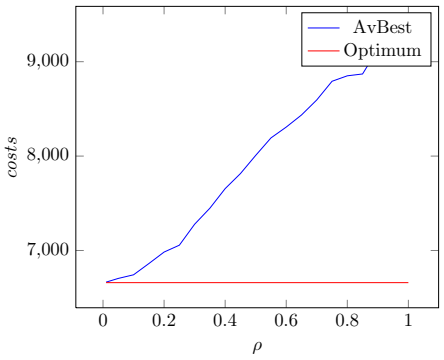


(f) Iteration 300: model

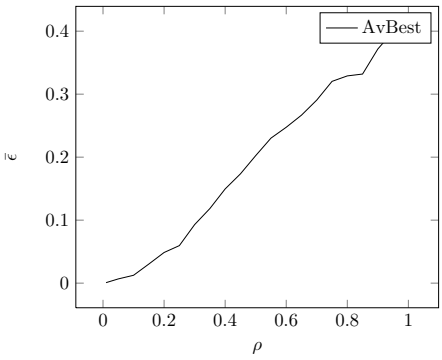
Figure 7.4: Observation of absorption of solutions and models (cont.)

Relation between ρ and the best found solution

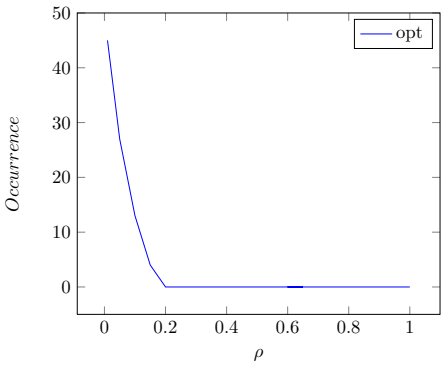
According to Table 7.2, we draw resp., a plot for ρ and the average best found cost $\text{Ave}_{f(\mathbf{X}^*)}$ in Figure 7.5 (a), a plot for ρ and the average relative error $\bar{\epsilon}$ in Figure 7.5 (b), and a plot for ρ and $Fre\{s^*\}$ (i.e. the number of trials in which optimal solutions occurred) in Figure 7.5 (c).



(a) for $\text{Ave}_{f(\mathbf{X}^*)}$



(b) for $\bar{\epsilon}$



(c) for $Fre\{s^*\}$

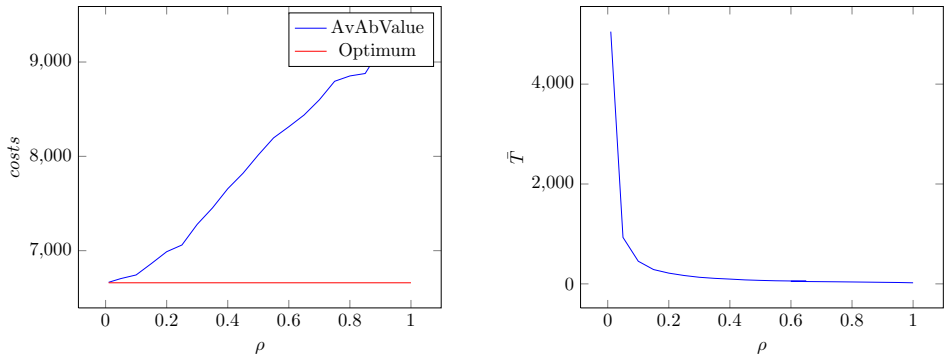
Figure 7.5: Relation of ρ and best found solution

In Figure 7.5 (a), the red straight line indicating the optimal value 6659.43. The blue line represents the average best found cost $\text{Ave}_{f(\mathbf{X}^*)}$. Figure 7.5 (a) and (b) clearly shows that as $\rho \to 0$, the best found solution will approximate optimal solutions. This means that we may improve the practical performance for an MBS by reducing the constant learning rate. It is interesting that the best solution cost almost *linearly* decreases as ρ increases.

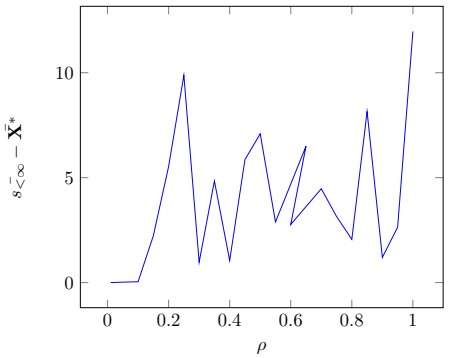
By 7.5 (c), we see that when $\rho \to 0$, $Fre\{s^*\} \to 50$. This experimentally demonstrates Theorem 5.7 c). In particular, we see that with a relatively large ρ (≥ 0.20), we are not able to see an optimal solution in the 50 trials. And when the ρ reaches the threshold 0.15 (see Table 7.2), $Fre\{s^*\}$ increases rapidly as ρ decreases.

Relation between ρ, $s_{<\infty}$ and the absorbing time

In the above, we have demonstrated a typical absorption behavior of the solutions and models. To give more insight to the qualities of absorbing solutions $s_{<\infty}$ and absorbing time, we draw resp., a plot for ρ and the average absorbing cost $\text{Ave}_{f(s_{<\infty})}$ in Figure 7.6 (a), a plot for ρ and the average absorbing time $\text{Ave}_{T_{ab}}$ in Figure 7.6 (b), and a plot for ρ and Diff in Figure 7.6 (c). In (a), the red line again indicates the optimum



(a) for $\text{Ave}_{f(s_{<\infty})}$ (b) for $\text{Ave}_{T_{ab}}$



(c) for Diff=$\text{Ave}_{f(s_{<\infty})} - \text{Ave}_{f(\mathbf{X}^*)}$

Figure 7.6: ρ and absorption

6659.43, the blue line is the average absorbing cost value $\text{Ave}_{f(s_{<\infty})}$. It is surprising that $\text{Ave}_{f(s_{<\infty})}$ also approximates optimum as $\rho \to 0$. This means that the absorbing solution may become an optimal solution if we employ a small enough ρ.

Absorbing time is the first iteration from which the sampling will be frozen at a fixed solution i.e. $s_{<\infty}$. Therefore, it directly affects the total number of different solutions the framework can visit. From Figure 7.6 (b), we see that the average absorption time $\text{Ave}_{T_{ab}}$ increases as $\rho \to 0$. So, decreasing ρ may increase the *search capacity* i.e. the total number of possible solutions the framework can visit. When $\rho < 0.15$ (see Table 7.2), $\text{Ave}_{T_{ab}}$ increases dramatically as ρ decreases. For $\rho \geq 0.15$, it changes moderately

$\rho_t \equiv \rho$	$f(s^*)$	$Fre(s^*)$	$\bar{\epsilon}$	$\text{Ave}_{f(\mathbf{X}^*)}$	$\text{Ave}_{T_{ab}}$	$\text{Ave}_{f(s_{<\infty})}$	Diff
1.00	6659.43	0	0.00556045	6696.46	3.78	6696.46	0.00
0.95	6659.43	7	0.00329671	6681.38	9.1	6681.38	0.00
0.90	6659.43	11	0.00281658	6678.19	9.74	6678.19	0.00
0.85	6659.43	10	0.00182641	6671.59	13.78	6671.59	0.00
0.80	6659.43	13	0.00203408	6672.98	15.42	6672.98	0.00
0.75	6659.43	17	0.00105385	6666.45	17.76	6666.45	0.00
0.70	6659.43	19	0.00182242	6671.57	19.3	6671.57	0.00
0.65	6659.43	24	0.000857031	6665.14	22.5	6665.14	0.00
0.60	6659.43	29	0.000586001	6663.33	25.6	6663.33	0.00
0.55	6659.43	32	0.00039527	6662.06	28.56	6662.06	0.00
0.50	6659.43	36	0.00025947	6661.16	31.94	6661.16	0.00
0.45	6659.43	41	0.000119201	6660.22	37.44	6660.22	0.00
0.40	6659.43	44	7.92956e-05	6659.96	43.58	6659.96	0.00
0.35	6659.43	46	5.53364e-05	6659.8	49.78	6659.8	0.00
0.30	6659.43	46	5.64757e-05	6659.81	62.7	6659.81	0.00
0.25	6659.43	49	1.42916e-05	6659.53	79.66	6659.53	0.00
0.20	6659.43	50	0	6659.43	105.18	6659.43	0.00
0.15	6659.43	50	0	6659.43	167.92	6659.43	0.00
0.10	6659.43	50	0	6659.43	347.36	6659.43	0.00

Table 7.3: Implementation results for greedy case

as ρ. And when $\rho > 0.20$, it is smaller than 200, see Table 7.2. Therefore, the search capacity is at most 2×10^4, i.e. the framework can visit at most (200×100) different tours, in this case. Compared to the size 6.88188×10^{42} of underlying feasible set, this number is extremely small. This may mean that the framework can be rather efficient when $\rho > 0.2$. However, we did not see an optimal solution in this case, see Table 7.2. To balance the efficiency and effectiveness, we would recommend to set $\rho = 0.05$ under which we observed optimal solutions in 27 trials and the average absorbing time is 930.44 i.e. the search capacity is 9.3×10^4, see Table 7.2.

Figure 7.6 (c) shows the difference between the average absorbing cost and the average best found cost i.e. $\text{Ave}_{f(s_{<\infty})} - \text{Ave}_{f(\mathbf{X}^*)}$. We see that this difference is generally not equivalent to 0. Therefore, the absorbing solution is generally not the same as the best solution found. However, when ρ small enough (< 0.2), the two costs may coincide. So, the two solutions may be the same for small ρ. This is an interesting phenomenon.

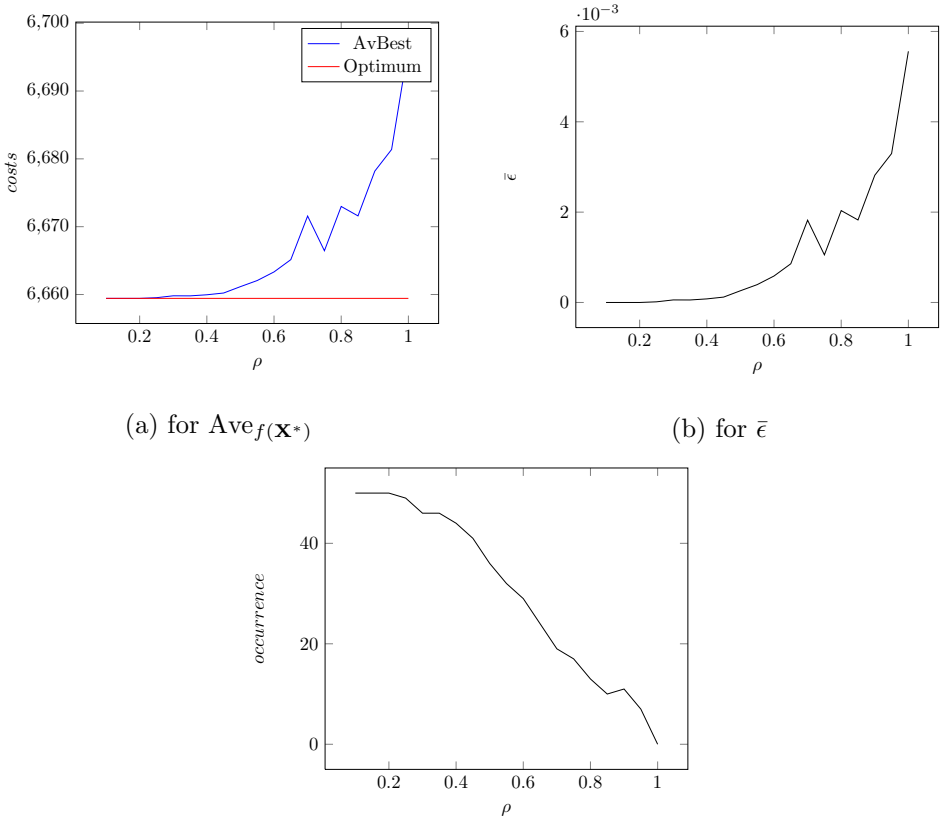
7.2.4 Under greedy feasibility construction

For the greedy feasibility construction, we take 19 different constant learning rates. For each learning rate, we also do 50 independent trials. Table 7.3 above shows the experimental results.

A significant difference with Table 7.2 is that the Column Diff in this case all become 0.

This means that under greedy feasibility construction, the absorbing solutions are more likely to be the best solution found. The second significant difference is Column $Fre(s^*)$ are almost filled by positive integers. This means the greedy feasibility construction significantly increases the occurring probability of optimal solutions.

Relation between ρ and best found solution



(a) for $\text{Ave}_{f(\mathbf{X}^*)}$

(b) for $\bar{\epsilon}$

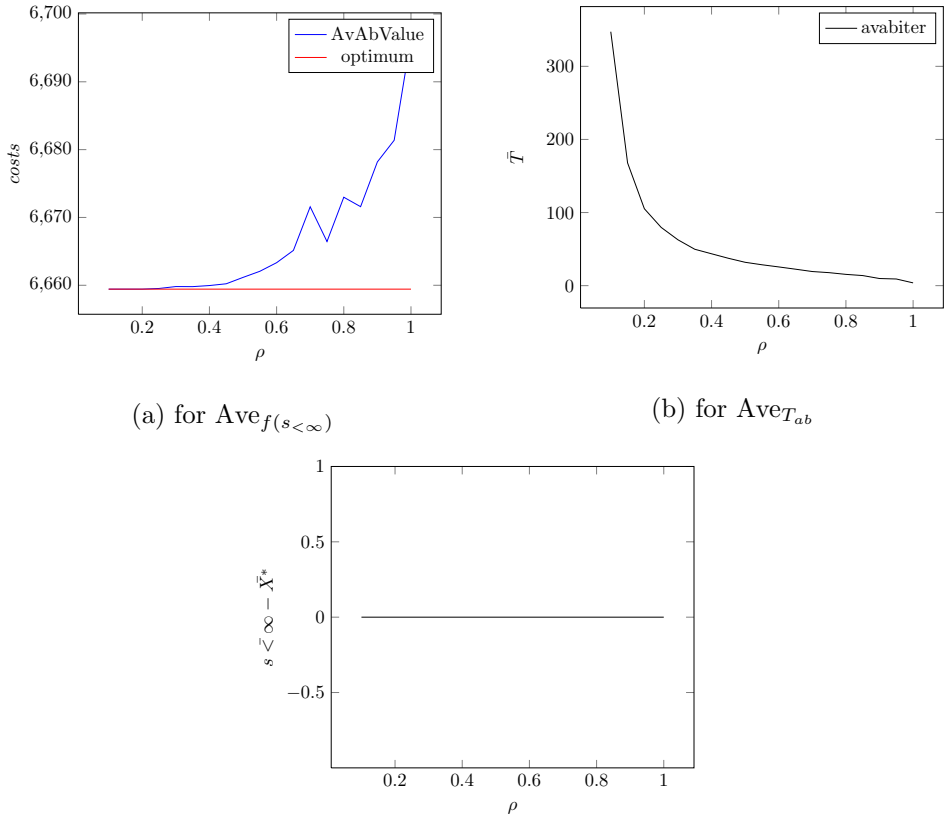
(c) for $Fre\{s^*\}$

Figure 7.7: ρ and best found solution for greedy case

Figure 7.7 is drawn according to Table 7.3. Picture (a) shows the plot of the average best found cost $\text{Ave}_{f(\mathbf{X}^*)}$. Picture (b) shows the plot for the relative error. Picture (c) shows the plot for the total number of trials among the 50 independent trials in which we observed an optimal solution.

According to Figure 7.7 (a)-(b), we see that under greedy feasibility construction, the average best found cost $\text{Ave}_{f(\mathbf{X}^*)}$ is very close to the optimum. In (a), the red horizontal line again indicates the optimum, and the blue line represents $\text{Ave}_{f(\mathbf{X}^*)}$. We see also that $\text{Ave}_{f(\mathbf{X}^*)}$ approximates optimum as ρ decreases. The maximal average relative error is

less than 6×10^{-3}, see (b). Moreover, by Figure 7.7 (c), we see that the probability $\mathbf{P}[\tau < \infty]$ may also approximate 1.0 as ρ decreases.



(a) for $\text{Ave}_{f(s_{<\infty})}$

(b) for $\text{Ave}_{T_{ab}}$

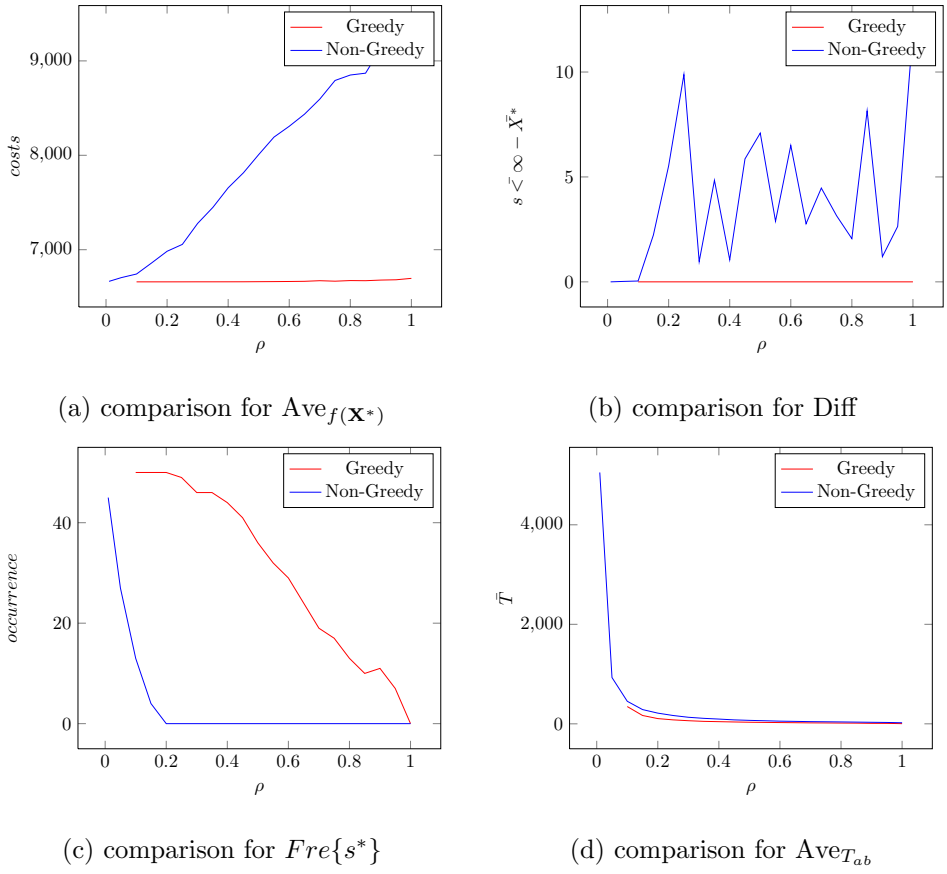
(c) for Diff=$\text{Ave}_{f(s_{<\infty})} - \text{Ave}_{f(\mathbf{X}^*)}$

Figure 7.8: ρ and absorption for greedy case

ρ and absorption

Figure 7.8 is also drawn according to Table 7.3. Picture (a) shows the plot for the average absorbing cost $\text{Ave}_{f(s_{<\infty})}$. Picture (b) shows the plot for the average absorbing time. Picture (c) shows the plot for the difference between the average best found cost and the average absorbing cost.

We can see from Figure 7.8 (a) that the average absorbing cost also decreases as ρ decreases. And it approximates the optimum when ρ is smaller enough. By (b), the average absorbing time increases as ρ decreases. So, in the greedy feasibility case, the search capacity also increases as ρ decreases. When ρ is small ($\rho < 0.2$), the time may also increase dramatically. It is interesting that in the greedy case, the absorbing cost is always the same as the best found cost, see (c). Therefore, the absorbing solution may be the best found solution.



(a) comparison for $\text{Ave}_{f(\mathbf{X}^*)}$

(b) comparison for Diff

(c) comparison for $Fre\{s^*\}$

(d) comparison for $\text{Ave}_{T_{ab}}$

Figure 7.9: Comparison

Compare with non-greedy case

According to Tables 7.2-7.3, we draw four pictures in Figure 7.9 for a comparison of greedy and non-greedy feasibility constructions. Picture (a) shows the plots for average best found costs in the two cases respectively, picture (b) shows the plots for the difference between average best found cost and average absorbing cost, picture (c) shows the plots for $Fre\{s^*\}$s, and picture (d) shows the plots for average absorbing times. In each picture, we use red line to indicate the greedy case, and blue line to indicate the non-greedy case.

By (a), we see that the greedy feasibility construction significantly improves the performance i.e. the average best found cost (red) in greedy case is significantly lower than average best found cost (blue) in the non-greedy case. In the picture, the red line is almost a horizontal line, since it is almost an optimum compared to the non-greedy case.

By (b), we see that the average absorbing cost is significantly different with the average best found cost in the non-greedy case (see blue line). However, the average absorbing cost is the same as the average best found cost in the greedy case (see red line). There-

fore, with greedy feasibility construction, the absorbing 'accuracy' is also significantly improved.

By (c), we see that the greedy feasibility construction may also increase the occurring probability of optimal solutions. However, when ρ is small enough, the advantage of greedy feasibility construction will be no longer significant.

By (d), the average absorbing time in greedy case is always smaller than in the non-greedy case. Therefore, greedy feasibility construction may be more efficient. To further compare the efficiencies of the two cases, we define the *relative efficiency* of greedy feasibility construction for a constant ρ as

$$\frac{\text{average absorbing time in non-greedy} - \text{average absorbing time in greedy}}{\text{average absorbing time in greedy}}.$$

For example, when $\rho = 1.0$, the average absorbing times are 21.56 in non-greedy case and 3.78 in greedy case, so the relative efficiency of greedy feasibility construction is

$$\frac{21.56 - 3.78}{3.78} \approx 4.7037,$$

see Tables 7.2-7.3 for the values. Note that, the average absorbing time is a very important index for the search capacity i.e. the total number of possible feasible solutions the framework can visit. Therefore, it may reflect the efficiency. By checking the two Tables more closely, we find that the relative efficiency may decrease as ρ decreases. We can see that when $\rho = 0.6$, the relative efficiency is about 1.1055, when $\rho = 0.10$, the relative efficiency is about 0.3048. Therefore, when ρ is smaller, the relative efficiency of the greedy feasibility construction may be no longer significant. So, we would recommend a relatively big ρ in the greedy case. To balance the efficiency and effectiveness, we would recommend $\rho = 0.5$ in the greedy case, by which we observed optimal solutions in 36 trials and the relative efficiency is 1.1803.

8 Summary and future work

In this Thesis, we investigated some important asymptotic properties for model-based heuristics in combinatorial optimization. To facilitate our analysis, we proposed a unified framework for model-based algorithms. In particular, we introduced a sampling mechanism called feasibility construction. It can unify the samplings in those model-based algorithms used in practice. More importantly, it can introduce some greediness and dependencies into the sampling.

To give some insight to the effectiveness of model-based search, we showed conditions for guaranteeing to find an optimal solutions. The conditions did not assume a particular structure of the underlying problem. Therefore, they may apply to all combinatorial optimization problems. For some test problems, we also proposed conditions for efficiently reaching an optimal solution.

In the asymptotic analysis of the underlying solutions, we found that the famous genetic drift may also hold in model based search. In particular, we found that the sampling in model-based search may stick on a fixed solution after finitely many iterations. Moreover, the solution freezing the sampling is generally of a high quality compared to those solutions seen in history.

In model based search, the asymptotic property of the models is of great importance. By our analysis, we found conditions which make the models converge to a limit concentrating on some optimal solution. However, for the memoryless case, we are not able to show a similar condition. But, our former work has shown that absorption of models and finite reachability are also compatible in this case.

By implementing the framework on a TSP instance, we are able to verify most of the findings. In particular, we found that the performance can be significantly improved if we use a 'good' greedy information in the feasibility construction. Moreover, we found that the absorbing solution can be improved if we use a small learning rate, and the absorbing time increases as the learning rate decreases. Therefore, higher learning rate may make the framework rather efficient. However, the quality of best solution found may be lower in this case.

As we mentioned, in the non-memory case, we are not able to show a condition which makes the models converge to a limit concentrating on optimal solutions. So, it should be an interesting topic for a future work. Note that this is an important issue for the cross entropy algorithm both in combinatorial optimization and in rare event simulation.

In Subsection 4.2.4, we said that the feasibility construction can introduce probabilistic dependencies into the sampling. The greedy information in TSP actually reflects some *priori* probabilistic dependencies. In the experimental study, we have seen that greedy feasibility construction indeed significantly improves the average performance. However, many practical problems may not have such a priori greedy information available. So,

another possibly future work is to learn *empirical* greedy information from a collection of best solutions found in history, and then incorporate it into the feasibility construction. This should be a topic closely related to the field of linkage learning.

In this Thesis, we investigated the runtime only on some rather simple problems. Due to simply structure, the results may not apply to practical problems. Therefore, a future work on runtime analysis should concern runtime results on some practical problems like TSP, scheduling, assignment etc. And the results may have significantly meaning in practice.

Appendix

A.1 Some definitions in graph theory

Graphs are fundamental structures in CO. In this section, we collect some elementary concepts and notations of graphs. All the materials we show here can be also found in some standard textbooks for CO, see e.g. Korte et al[KV02] Chapter 2.

Given V a *finite* set, and $E \subseteq \{(c,d)|c \in V \text{ and } d \in V\}$. Then, we call the pair $G = (V, E)$ as a *graph*, and each element in V as a *node* or *vertex*. Furthermore, if E satisfies the condition that,

$$\forall c, d \in V,\ (c,d) \in E \iff (d,c) \in E,$$

we call G as an *undirected* graph, otherwise G is called as a *directed* graph. For an undirected graph G, we call each element in E as an *edge*. For a directed graph G, we call each element in E as an *arc*. Let $e = (v_1, v_2)$ be an edge (an arc in the directed case correspondingly), then we say that vertices v_1 and v_2 are *adjacent*. v_1 is a *neighbor* of v_2 (and vice verse). v_1 and v_2 are called *endpoints* of e. In the directed case, we say that e *leaves* v_1 and *enters* v_2. Two edges or arcs which share at least one endpoint are *adjacent*. And if a vertex is an endpoint of an edge, we say that it is *incident* with the edge. Then, for an arbitrary vertex, we can define its *degree* as the number of edges (or arcs) which are incident to it. And in the directed case, we can define the *out-degree* of a vertex as the number of arcs leaving it and the *in-degree* as the number of arcs entering it.

A *walk* W on a graph $G = (V, E)$ is a sequence $v_1, e_1, v_2, \ldots, v_k, e_k, v_{k+1}$ with

- $v_i \in V$ for all $i = 1, \ldots, k+1$,
- $e_i = (v_i, v_{i+1}) \in E$ for all $i = 1, \ldots, k$,
- $e_i \neq e_j$ for all $i \neq j$ and $i, j \in \{1, \ldots, k\}$.

Obviously, a walk can also be represented as a sequence of vertices or a sequence of edges (arcs). And if $W = (v_1, e_1, v_2, \ldots, v_k, e_k, v_{k+1})$ is a walk, then we call v_1 and v_{k+1} as the *endpoints* in the undirected case, we call v_1 as the *start point* and v_{k+1} as the *end point* in the directed case. We say a walk is *closed* if the endpoints (start and end point in the directed case) are the same. A *path* is a walk with different vertices, i.e. a walk $(v_1, e_1, v_2, \ldots, v_k, e_k, v_{k+1})$ with $v_i \neq v_j$ for $i \neq j$. And a *circuit* or *circle* is a closed path, i.e. the underlying walk is closed.

A *subgraph* $G_1 = (V_1, E_1)$ of $G = (V, E)$ is a graph such that $V_1 \subseteq V$ and $E_1 \subseteq E$. And if $V_1 = V$, we say that G_1 is a *spanning* subgraph of G. Then a *spanning walk* (path and circuit resp.) is a walk which contains all possible vertices on G, and a *Hamiltonian circle* or *tour* is a spanning circuit on G.

Assume $G = (V, E)$ is a graph, and $w : E \mapsto \mathbb{R}$ is a real function defined on the set of edges or arcs, then we call the triple (V, E, w) as a *weighted* graph, and w as a *weight function* of graph G.

Suppose $G = (V, E)$ is an undirected graph, we say it is *connected* if for any two vertices v_1 and v_2, we can find a path with v_1 and v_2 as endpoints. And we say that G is *fully* or *completely connected* if and only if for any two vertices a and b in V, we have $(a, b) \in E$. Evidently, fully connected implies connected.

In undirected case, a *forest* is a graph without circuits. And a connected forest is called a *tree*. A vertex in a tree with degree 1 is called a *leaf*. And in the directed case, a *root* on a tree is a vertex which has 0 in-degree. Then a *branch* can be formally defined as a path which leaves the root and enters a leaf.

A.2 Shannon Entropy, K-L divergence and importance sampling

This section proposes some terminologies in information theory. Elements in this section can also be found in Cover et al[CT12] and Rubinstein et al[RK04].

In information theory, Shannon entropy or entropy is the most celebrated measure of uncertainty. Let X be a random variable with density f. Then the *Shannon entropy* or *entropy* of X is defined as

$$\mathfrak{E}(X) := \mathbf{E}\Big[\ln\big(\frac{1}{f(X)}\big)\Big] = -\mathbf{E}\big[\ln f(X)\big] = -\int f(x)\ln f(x)dx.$$

Obviously, if X concentrates on one value, i.e. $\mathbf{P}(X = a) = 1$ for some constant value a, then $\mathfrak{E}(X) = 0$. And if X is uniform distributed, $\mathfrak{E}(X)$ reaches maximum.

Similarly, we can define the *joint* or *total entropy* $\mathfrak{E}(X_1, \ldots, X_m)$ of random variables $X_1, \ldots, X_m$ with joint density $f(x_1, \ldots, x_m)$ as

$$\mathfrak{E}(X_1, X_2, \ldots, X_m) := -\int f(x_1, x_2, \ldots, x_m)\ln f(x_1, x_2, \ldots, x_m)dx_1dx_2\cdots dx_m.$$

It measures the 'joint' or 'total' uncertainty of random variables $X_1, \ldots, X_m$.

For two random variables X_1 and X_2 with joint density f, we can reasonably define the *conditional entropy* of X_1 given X_2 as

$$\mathfrak{E}(X_1|X_2) := \mathfrak{E}(X_1, X_2) - \mathfrak{E}(X_2) = -\mathbf{E}\Big[\ln\frac{f(X_1, X_2)}{f_{X_1}(X_1)}\Big],$$

where $f_{X_1}(\cdot)$ is the marginal density of X_1. Note that the conditional entropy measures the remaining uncertainty when X_2 is given.

To measure the mutual dependency of two random variables, we define the *mutual information* $\mathfrak{I}(X_1, X_2)$ of two random variables with joint density f as

$$\mathfrak{I}(X_1, X_2) := \mathfrak{E}(X_1) + \mathfrak{E}(X_2) - \mathfrak{E}(X_1, X_2) = \mathfrak{E}(X_1) - \mathfrak{E}(X_1|X_2) = \mathfrak{E}(X_2) - \mathfrak{E}(X_2|X_1).$$

Two basic facts about mutual information is that

- $\mathfrak{I}(X_1, X_2) = \mathfrak{I}(X_2, X_1)$,
- $\mathfrak{I}(X_1, X_2) = 0 \iff X_1$ and X_2 are probabilistically independent.

The Kullback-Leibler divergence is a 'metric' of densities. It can be used to measure the 'distance' of two probability densities. Let g and f be two densities with respect to a measure μ on a space $\mathbf{X}$. The *Kullback-Leibler divergence* (K-L divergence) of f respect to g is

$$\mathfrak{D}(f; g) := \mathbf{E}_g\Big[\ln\frac{g(X)}{f(X)}\Big] = \int -g(x)\ln f(x)dx - \int -g(x)\ln g(x)dx.$$

Where $\mathbf{E}_g(\cdot)$ indicates that the expectation is respect to density g. It is easy to check that $\mathfrak{D}(f; g) \geq 0$, and $\mathfrak{D}(f; g) = 0$ if and only if $f = g$, μ- almost surely. $\mathfrak{D}(\cdot; \cdot)$ is also called *cross entropy* or *relative entropy*.

A.3 Linear programming and integer programming

Linear programming optimization (LPO, see [KV02] p. 49) concerns solving problems like

$$\begin{aligned} &\textbf{minimizing:} && c^T x, \\ &\textbf{subject to:} && Ax \leq b, \\ & && x \in \mathbb{R}^n, \end{aligned}$$

where matrix $A \in \mathbb{R}^{m \times n}$, column vector $b \in \mathbb{R}^m$ and column vector $c \in \mathbb{R}^n$ are constant. A LPO instance is called a *linear program* (LP). And most of the COPs can be formulated as LPs.

Integer programming optimization (IPO, see [KV02] p. 91) can be seen as a subtopic of LPO which concerns solving the LPs with $A \in \mathbb{Z}^{m \times n}$, $b \in \mathbb{Z}^m$ and $x, c \in \mathbb{Z}^n$. A more restrictive research topic is *zero-one integer programming optimization* which concentrates on LPs with $A \in \{0,1\}^{m \times n}$, $b \in \{0,1\}^m$ and $x, c \in \{0,1\}^n$. Obviously, zero-one integer programming is a subtopic of CO. Actually, due to the convexity of feasible regions, LPO can also be seen as a subtopic of CO.

A.4 Elements about probability theory and stochastic process

Here, we collect some elements about probability theory and stochastic process, see [Cin75] and [Dur10] for a further reference.

In probability theory, we call a countable or uncountable set Ω as a *sample space.* Then an element in Ω is called as an *outcome,* and a subset of Ω is called as a *random event*.

Let $\mathcal{B}$ be a collection of random events on a sample space Ω. We say that $\mathcal{B}$ is an *σ-algebra* on Ω if and only if

i) it contains complete event i.e. $\Omega \in \mathcal{B}$,

ii) it is closed under complement i.e. $A \in \mathcal{B} \ \Rightarrow \ \Omega - A \in \mathcal{B}$,

iii) it is closed under infinite intersection i.e. $\forall n \in \mathbb{N}\ A_n \in \mathcal{B}, \ \Rightarrow \ \bigcup_{n\in\mathbb{N}} A_n \in \mathcal{B}$.

And we call pair $\big(\Omega, \mathcal{B}\big)$ as a *measurable space* if $\mathcal{B}$ is a σ-algebra on Ω. For a measurable space $\big(\Omega, \mathcal{B}\big)$, we call each event in $\mathcal{B}$ as a *$\mathcal{B}$-measurable set.* A typical measurable space is the so-called *Borel space.* Let $\mathbb{R}^n = \mathbb{R} \times \cdots \times \mathbb{R}$ be the Euclidean space of dimension n. And $\mathfrak{B}_n$ denotes the smallest σ-algebra which contains all of open sets in $\mathbb{R}^n$. Then we call $(\mathbb{R}^n, \mathfrak{B}_n)$ as the n-dimensional Borel space.

Let $\big(\Omega, \mathcal{B}\big)$ be a measurable space. Let $\mathbf{P} : \mathcal{B} \mapsto [0, 1]$ be a set function such that

i) $\mathbf{P}(\Omega) = 1$,

ii) let $\{A_n\}_{n\in\mathbb{N}}$ be a sequence in $\mathcal{B}$ s.t. $A_n \cap A_m = \emptyset$ for $n \neq m$, then

$$\mathbf{P}\Big[\ \bigcup_{n\in\mathbb{N}} A_n\ \Big] = \sum_{n\in\mathbb{N}} \mathbf{P}\big[A_n\big],$$

then $\mathbf{P}$ is called a probabilistic measure on $\big(\Omega, \mathcal{B}\big)$, and triple $\big(\Omega, \mathcal{B}, \mathbf{P}\big)$ is called a probability space.

Given a probability space $\big(\Omega, \mathcal{B}, \mathbf{P}\big)$. The probability of a random event $A \subseteq \Omega$ is defined as

$$\mathbf{P}(A) := \inf\big\{\mathbf{P}(B) \ \big|\ B \in \mathcal{B},\ A \subseteq B\big\},$$

i.e. the outer measure of A under probabilistic measure $\mathbf{P}$. Note that measurable set must be a random event, but a random event may not be a measurable set. We say that two random events A_1 and A_2 are *independent* if

$$\mathbf{P}\big[A_1 \cap A_2\big] = \mathbf{P}\big[A_1\big] \cdot \mathbf{P}\big[A_2\big].$$

Suppose $\mathbf{P}(A) > 0$ for $A \in \mathcal{B}$. Then the probability of a random event B *conditioned on* A is defined as

$$\mathbf{P}\big[B \ \big|\ A\big] := \frac{\mathbf{P}\big[A \cap B\big]}{\mathbf{P}\big[A\big]}.$$

A very useful formula about conditional probability is

$$\mathbf{P}\big[\bigcap_{t\in\mathbb{N}} A_t\big] = \mathbf{P}[A_0]\prod_{t=1}^{\infty}\mathbf{P}\big[A_t \mid \forall m \leq t-1 \quad A_m\big]$$

which is a standard technique for showing the probability of chain events.

Suppose $\big(\Omega, \mathcal{B}\big)$ is a measurable space. Then we call a function $X : \Omega \mapsto \mathbb{R}^n$ as an n-dimensional *measurable function* if for all $B \in \mathfrak{B}_n$, $X^{-1}(B) \in \mathcal{B}$. In probability theory, we often call a measurable function X as a *random variable.* And $E = X(\Omega) \subseteq \mathbb{R}^n$ is called the *state space* of that random variable. For a Borel measurable set $B \in \mathfrak{B}_n$, and a probabilistic measure $\mathbf{P}$, we write

$$\mathbf{P}\big[X \in B\big] := \mathbf{P}\big\{\omega \in \Omega \mid X(\omega) \in B\big\}.$$

Let X_1 and X_2 be two random variables, then we say that they are independent if and only if

$$\mathbf{P}\big[X_1 \in B_1,\ X_2 \in B_2\big] = \mathbf{P}\big[X_1 \in B_1\big] \cdot \mathbf{P}\big[X_2 \in B_2\big]$$

for all $B_1, B_2 \in \mathfrak{B}_n$.

Let X is an 1-dimensional random variable on a probability space $\big(\Omega, \mathcal{B}, \mathbf{P}\big)$. Then we call the function

$$\mathbf{F}(x) := \mathbf{P}\big[X \leq x\big] \quad \text{for all } x \in \mathbb{R}$$

as the *distribution function* of X. And if the state space E of X is countable, say $\{x_i \in \mathbb{R} \mid i \in \mathbb{N}\}$, its distribution function may be described by a vector $\big(\mathbf{P}[X = x_i]\big)_{i\in\mathbb{N}}$ i.e.

$$\mathbf{F}(x) = \sum_{x_i \leq x, x_i \in E} \mathbf{P}[X = x_i].$$

Suppose

$$\Big\{\big(\Omega_\theta, \mathcal{B}_\theta, \mathbf{P}_\theta\big)\Big\}_{\theta\in\Theta}$$

is a family of probability spaces. We write

$$\prod_{\theta\in\Theta}\Omega_\theta := \big\{(\omega_\theta)_{\theta\in\Theta} \mid \omega_\theta \in \Omega_\theta, \theta \in \Theta\big\}$$

as the product sample space. Let $\prod_{\theta\in\Theta}\mathcal{B}_\theta$ be the smallest σ-algebra on $\prod_{\theta\in\Theta}\Omega_\theta$ which contains the collection

$$\big\{C(A_{\theta_1}, \ldots, A_{\theta_n}) \mid A_{\theta_1} \in \mathcal{B}_{\theta_1}, \ldots, A_{\theta_n} \in \mathcal{B}_{\theta_n}, n \in \mathbb{N}\big\},$$

where

$$C(A_{\theta_1}, \ldots, A_{\theta_n}) := \big\{(\omega_\theta)_{\theta\in\Theta} \mid \omega_{\theta_1} \in A_1, \ldots, \omega_{\theta_n} \in A_{\theta_n}\big\} \subseteq \prod_{\theta\in\Theta}\Omega_\theta.$$

And define $\prod_{\theta\in\Theta}\mathbf{P}_\theta$ as the unique probabilistic measure on $\big(\prod_{\theta\in\Theta}\Omega_\theta, \prod_{\theta\in\Theta}\mathcal{B}_\theta\big)$ such that

$$\prod_{\theta\in\Theta}\mathbf{P}_\theta\big(C(A_{\theta_1}, \ldots, A_{\theta_n})\big) = \prod_{i=1}^{n}\mathbf{P}_{\theta_i}(A_{\theta_i}) \quad \text{for all } A_{\theta_1} \in \mathcal{B}_{\theta_1}, \ldots, A_{\theta_n} \in \mathcal{B}_n \text{ with } n \in \mathbb{N}.$$

Then the probability space $\big(\prod_{\theta\in\Theta}\Omega_\theta, \prod_{theta\in\Theta}\mathcal{B}_\theta, \prod_{\theta\in\Theta}\mathbf{P}_\theta\big)$ is called a *product probability space.*

Let $\mathfrak{M}$ be the collection of all possible 1-dimensional random variables on a probability space $\big(\Omega, \mathcal{B}, \mathbf{P}\big)$. Then we say that a function $\mathbf{E} : \mathfrak{M} \mapsto \mathbb{R}$ is the *expectation* operator if and only if it satisfies

i) for all $A \in \mathcal{B},\ \mathbf{E}\big(\mathbb{1}_A\big) = \mathbf{P}(A)$,
ii) if $X_n \to X$ as $n \to \infty$ and $X_n \in \mathfrak{M}$ for all $n \in \mathbb{N},$ then $\mathbf{E}(X_n) \to \mathbf{E}(X)$ as $n \to \infty$,
iii) for all $c \in \mathbb{R}$ and $X, Y \in \mathfrak{M},\ \mathbf{E}(c \cdot X) = c \cdot \mathbf{E}(X)$ and $\mathbf{E}(X + Y) = \mathbf{E}(X) + \mathbf{E}(Y)$,

where observe that $\mathbb{1}_A(\omega) \in \mathfrak{M}$. And note that $\mathfrak{M}$ is closed under limit. And furthermore, if Ω is countable, then

$$\mathbf{E}(X) = \sum_{\omega\in\Omega} X(\omega)\mathbf{P}\{\omega\} \quad \text{for all } X \in \mathfrak{M}.$$

Suppose $\big(\Omega, \mathcal{B}, \mathbf{P}\big)$ is a probability space. Let $(X_t)_{t\in\Gamma}$ be a sequence of random variables with a common state space E on it. Then we say $(X_t)_{t\in\Gamma}$ is a *stochastic process* with state space E. And if Γ is countable, we say that $(X_t)_{t\in\Gamma}$ is *discrete* parameter process.

Let $(X_t)_{t\in\mathbb{N}}$ be a discrete process on $\big(\Omega, \mathcal{B}, \mathbf{P}\big)$. Then we say that it is a *Markov chain* if and only if

$$\mathbf{P}\big[X_{t+1} \in B \;\big|\; X_t \in B_t, X_{t-1} \in B_{t-1}, \ldots, X_0 \in B_0\big] := \mathbf{P}\big[X_{t+1} \in B \;\big|\; X_t \in B_t\big]$$

for all $t \in \mathbb{N}$, all $B \in \mathfrak{B}_n$, and also all $B_i \in \mathfrak{B}_n$ for $i = 0, 1, \ldots, t$.

A.5 Useful notations in runtime analysis

Suppose f and g are two positive functions on natural numbers.

Big O notation:

We write $f \in O(g)$ if and only if

$$\exists a > 0 \; \exists N \in \mathbb{N} \; \forall n \geq N \quad f(n) \leq a \cdot g(n).$$

Big O notation is used to show that f is *bounded above* by g asymptotically. Obviously, $f \in O(g)$ if and only if $\lim_{n\to\infty} \sup \; f(n)/g(n) < \infty$.

Small o notation:

We write $f \in o(g)$ if and only if

$$\forall a > 0 \; \exists N \in \mathbb{N} \; \forall n \geq N \quad f(n) \leq a \cdot g(n).$$

$f \in o(g)$ means that f *is dominated* by g asymptotically. And $f \in o(g)$ if and only if $\lim_{n\to\infty} f(n)/g(n) = 0$.

Big Ω notation:

We write $f \in \Omega(g)$ if and only if

$$\exists a > 0 \; \exists N \in \mathbb{N} \; \forall n \geq N \quad f(n) \geq a \cdot g(n).$$

So $f \in \Omega(g)$ means that f is *bounded below* by g asymptotically. And $f \in \Omega(g)$ if and only if $g \in O(f)$.

Small ω notation:

We write $f \in \omega(g)$ if and only if

$$\forall a > 0 \; \exists N \in \mathbb{N} \; \forall n \geq N \quad f(n) \geq a \cdot g(n).$$

$f \in \omega(g)$ if and only if f *dominates* g asymptotically.

Big Θ notation:

We write $f \in \Theta(g)$ if and only if $f \in O(g) \cap \Omega(g)$, namely f is *bounded both below and above* by g asymptotically.

$\sim$ notation:

We write $f \sim g$ if and only if they are asymptotically equivalent i.e.

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 1.$$

A.6 Application of cross entropy algorithm in rare event simulation

CE is initially motivated for rare event simulation (RES). As an additional reference on CE, we introduce this in the present Section. For a detailed reference on RES, see the book [RT$^+$09].

A.6.1 A brief introduction to rare event simulation

Generally, a *rare event* is an event which occurs with an extremely small probability. For example, a catastrophic failure for a civil aircraft during a flight is generally a rare event. As a security or reliability assessment, we may need to calculate or estimate occurring probabilities of some 'unpleasant' rare events for a complex system in practice.

Typically, we may need to calculate a probability

$$\mathbf{P}(\mathfrak{A}) = \int_{(x_1,\ldots,x_n)\in\Omega} \mathbb{1}_{\{f(y_1,\ldots,y_n)\leq\gamma\}}(x_1,\ldots,x_n)g(x_1,\ldots,x_n)dx_1\cdots dx_n \tag{1}$$

where $n \in \mathbb{N}$ is a fixed integer and $\mathfrak{A}$ is a rare event of form

$$\left[\ f\big(X_1,\ldots,X_n\big) \leq \gamma\ \right] \tag{2}$$

for a fixed *threshold value* $\gamma \in \mathbb{R}$, random variables $X_1,\ldots,X_n$ with joint density $g(x_1,\ldots,x_n)$ and *state space* Ω, and $f : \Omega \mapsto \mathbb{R}$ is a *performance function.* Here, an element $x = (x_1,\ldots,x_n) \in \Omega$ corresponds to a state for a system, and a failure occurs if the system is in a state $x \in \Omega$ with performance value $f(x) \leq \gamma$.

In theory, we can estimate (1) by the so-clalled crude monte-carlo method (see Appendix A. 2 or [RT$^+$09] pp. 2-4), and the corresponding estimator is

$$\widehat{\mathbf{P}(\mathfrak{A})} = \frac{1}{N}\sum_{j=1}^{N} \mathbb{1}_{(-\infty,\gamma]}\Big(f\big(\mathbf{X}^{(j)}(1),\ldots,\mathbf{X}^{(j)}(n)\big)\Big), \tag{3}$$

where $\mathbf{X}^{(1)} = \big(\mathbf{X}^{(1)}(1),\ldots,\mathbf{X}^{(1)}(n)\big),\ldots,\mathbf{X}^{(N)} = \big(\mathbf{X}^{(N)}(1),\ldots,\mathbf{X}^{(N)}(n)\big)$ are *i.i.d* from density g. Estimator (3) has a coefficient of variation (c.v.)

$$\frac{\sqrt{\mathbf{Var}\big[\widehat{\mathbf{P}(\mathfrak{A})}\big]}}{\mathbf{E}\big[\widehat{\mathbf{P}(\mathfrak{A})}\big]} = \sqrt{\frac{\big(1-\mathbf{P}(\mathfrak{A})\big)}{N\cdot\mathbf{P}(\mathfrak{A})}}.$$

When $\mathbf{P}\big(\mathfrak{A}\big)$ is not too small, we can make the above c.v. in a tolerant level by taking a sufficiently large N. However, the $\mathfrak{A}$ here is a rare event. To get an effective approximation to (1), we may need an extremely huge N. For example, $\mathbf{P}(\mathfrak{A})$ may be smaller than 10^{-9}, then we have to take an $N \geq 10^{13}$ so as to make the c.v. smaller than 0.01. Obviously, this is prohibitive in practice.

In RES, we concern how to estimate (1) or simulate $\mathfrak{A}$ effectively for the case that $\mathbf{P}\big(\mathfrak{A}\big)$ is extremely small. *Importance sampling* and *splitting techniques* are the most frequently used approaches in RES. Here, we shall concentrate only on importance sampling. For the splitting techniques, see [RT$^+$09] Chapter 2 of Part I for an introduction.

A.6.2 Importance Sampling

Let $x = (x_1, \ldots, x_n), y = (y_1, \ldots, y_n)$ and $X = (X_1, \ldots, X_n)$. Then, (1) can be written as

$$\begin{aligned} \mathbf{P}(\mathfrak{A}) &= \int_{x\in\Omega} \mathbb{1}_{\{f(y)\le\gamma\}}(x)g(x)dx = \int_{x\in\Omega} \mathbb{1}_{\{f(y)\le\gamma\}}(x)\frac{g(x)}{h(x)}h(x)dx \\ &= \mathbf{E}_h\Big[\mathbb{1}_{\{f(y)\le\gamma\}}(X)\frac{g(X)}{h(X)}\Big], \end{aligned} \tag{4}$$

where h is an arbitrary density on Ω with $h(x) = 0 \Rightarrow g(x) = 0$ for any $x \in \Omega$, and notation $\mathbf{E}_h[\cdot]$ indicates the expectation is taken under density h. Of cause, (4) again employs the convention $\frac{0}{0} = 0$. (4) is called a *change of measure* (CoM) with density h. With that CoM, we may estimate (1) as

$$\widehat{\mathbf{P}(\mathfrak{A})|_h} = \frac{1}{N}\sum_{j=1}^{N} \mathbb{1}_{(-\infty,\gamma]}\big(f(\mathbf{X}^{(j)})\big)\frac{g(\mathbf{X}^{(j)})}{h(\mathbf{X}^{(j)})} \tag{5}$$

with $\mathbf{X}^{(1)}, \ldots, \mathbf{X}^{(N)}$ *i.i.d* from $h(\cdot)$. Obviously, the c.v. of (5) is

$$\frac{\sqrt{\mathbf{Var}_h\big[\widehat{\mathbf{P}(\mathfrak{A})|_h}\big]}}{\mathbf{E}_h\big[\widehat{\mathbf{P}(\mathfrak{A})|_h}\big]} = \frac{\sqrt{\mathbf{Var}_h\big[\widehat{\mathbf{P}(\mathfrak{A})|_h}\big]}}{\mathbf{P}(\mathfrak{A})} = \frac{\sqrt{\mathbf{E}_h\Big[\mathbb{1}_{\{f(y)\le\gamma\}}(X)\frac{g(X)}{h(X)}\Big]^2 - \Big[\mathbf{P}(\mathfrak{A})\Big]^2}}{\sqrt{N}\cdot\mathbf{P}(\mathfrak{A})}. \tag{6}$$

By the well-known Jensen's inequality [Jen06], whenever h equals to

$$g^*(x) = \frac{\mathbb{1}_{\{f(y)\le\gamma\}}(x)g(x)}{\int_{z\in\Omega}\mathbb{1}_{\{f(y')\le\gamma\}}(z)g(z)dz} = \frac{\mathbb{1}_{\{f(y)\le\gamma\}}(x)g(x)}{\mathbf{P}(\mathfrak{A})} \quad \text{for all } x \in \Omega, \tag{7}$$

the c.v. in (6) reaches the minimal value 0. We often call g^* as the *zero variance change of measure*.

Ideally, we may use g^* as a CoM in (4). Then the resulted estimator in (5) shall have a 0 c.v.. Unfortunately, this is infeasible in practice, since g^* has the unknown quantity $\mathbf{P}(\mathfrak{A})$ as dominator. A clever alternative approach was proposed by R. Y. Rubinstein in [Rub97]. It first specifies a family of densities $\mathbb{P}_{IS} = \{g(\cdot;\theta) : \theta \in \Theta\}$ such that we can easily draw samples from each $g(\cdot;\theta)$. Then, it employs a search in $\mathbb{P}_{IS}$ for a density which 'mimics' g^* best. Finally, it uses the 'optimal' density found in $\mathbb{P}_{IS}$ as a CoM in (4), and estimates $\mathbf{P}(\mathfrak{A})$ by the corresponding estimator in (5). Since the density found by the search may approximate g^* well, the resulted estimator can be effective enough. To achieve the idea, a metric or criterion for determining the 'similarity' or 'distance' between densities is needed. R. Y. Rubinstein proposed two possible criterion in [Rub97] and [Rub99], namely *variance minimization* (VM) and *K-L divergence* (Kullback-Leibler distance or divergence, see A. 2 in the Appendix). With a fixed criterion, the search generally results in an iteratively stochastic procedure in Θ. Below, we shall employ K-L divergence as an example.

A.6.3 Application of cross entropy in rare event simulation

Now, we fix a family $\mathbb{P}_{IS} = \{g(\cdot;\theta)|\theta \in \Theta\}$ such that each $g(\cdot;\theta)$ is a density function on the state space Ω for $\theta \in \Theta$, and there is a $\theta_0 \in \Theta$ with $g(\cdot) = g(\cdot;\theta_0)$.

Generally, the K-L divergence of a density h to another density k is

$$\mathfrak{D}\big(h;k\big) = -\int_{x\in\Omega} k(x)\ln h(x)dx + \int_{x\in\Omega} k(x)\ln k(x)dx.$$

It has properties as following:

- $\mathfrak{D}\big(h;k\big) \geq 0$ for all h, k;
- $\mathfrak{D}\big(h;k\big) = 0$ if and only if $f = g$ holds almost surely.

See A. 2 in the Appendix for an explanation. Therefore, we can find out a best approximation $g(\cdot;\theta^*)$ with $\theta^* \in \Theta$ for g^* by solving

$$\min_{\theta\in\Theta} \mathfrak{D}\big(g(\cdot;\theta);g^*\big) = \min_{\theta\in\Theta}\Big\{-\int_{x\in\Omega} g^*(x)\ln g(x;\theta)dx + \int_{x\in\Omega} g^*(x)\ln g^*(x)dx\Big\}.$$

Observe that

$$\int_{x\in\Omega} g^*(x)\ln g^*(x)dx$$

is a constant. By (7), we may equivalently solve

$$\begin{aligned}\min_{\theta\in\Theta} -\int_{x\in\Omega} g^*(x)\ln g(x;\theta)dx &\iff \max_{\theta\in\Theta}\int_{x\in\Omega} g^*(x)\ln g(x;\theta)dx \\ &\iff \max_{\theta\in\Theta}\int_{x\in\Omega}\frac{\mathbb{1}_{\{f(y)\leq\gamma\}}(x)g(x)}{\mathbf{P}(\mathfrak{A})}\ln g(x;\theta)dx\end{aligned}$$

instead. Note that $\mathbf{P}(\mathfrak{A})$ is a constant, we can solve

$$\max_{\theta\in\Theta}\int_{x\in\Omega}\mathbb{1}_{\{f(y)\leq\gamma\}}(x)g(x)\ln g(x;\theta)dx \iff \max_{\theta\in\Theta}\mathbf{E}_{\theta_0}\Big[\mathbb{1}_{\{f(y)\leq\gamma\}}(X)\ln g(X;\theta)\Big] \tag{8}$$

for an optimal $\theta^* \in \Theta$, where recall that $g(x) = g(x;\theta_0)$ and $\mathbf{E}_{\theta_0}[\cdot]$ is the expectation under density $g(\cdot;\theta_0)$.

In practice, it is often impossible for us to solve (8) deterministically. In [Rub99], an iteratively stochastic search procedure was proposed for solving (8) . And with a minor change, the procedure then become the CE algorithm on p. 19. This is also why we say that CE algorithm is initially motivated for RES. Formally, the procedure results in a stochastic process $(\theta_t;\gamma_t;\mathbf{X}_t)_{t=0,1,2,\ldots}$ with

- $\mathbf{X}_t = \big(\mathbf{X}_t^{(1)},\ldots,\mathbf{X}_t^{(N)}\big)$ is a random sample of a specified size N from density $g(\cdot;\theta_t)$, where each $\mathbf{X}_t^{(j)} = (\mathbf{X}_t^{(j)}(1),\ldots,\mathbf{X}_t^{(j)}(n)) \in \Omega$ for $j = 1,\ldots,N$;
- $\gamma_t = f(\mathbf{X}_t^{(m_{\lfloor\alpha\cdot N\rfloor})})$ where $\alpha \in (0,1)$ a not very small constant, and $\mathbf{X}_t^{(m_1)},\ldots,\mathbf{X}_t^{(m_N)}$ is the increasing order of $\mathbf{X}_t$ according to the performance function f i.e.

$$f(\mathbf{X}_t^{(m_1)}) \leq f(\mathbf{X}_t^{(m_2)}) \leq \cdots \leq f(\mathbf{X}_t^{(m_N)}),$$

- $\theta_0 \in \Theta$ is fixed such that $g(\cdot) = g(\cdot;\theta_0)$ and $\theta_{t+1} \in \Theta$ is a solution of

$$\max_{\theta\in\Theta} \frac{1}{N}\sum_{j=1}^{N} \mathbb{1}_{\{f(y)\leq\gamma_t\}}(\mathbf{X}^{(j)})\frac{g(\mathbf{X}_t^{(j)};\theta_0)}{g(\mathbf{X}_t^{(j)};\theta_t)}\ln g(\mathbf{X}_t^{(j)};\theta). \tag{9}$$

Note that (9) is a stochastic counterpart of

$$\max_{\theta\in\Theta}\mathbf{E}_{\theta_t}\Big[\mathbb{1}_{\{f(y)\leq\gamma_t\}}(X)\frac{g(X;\theta_0)}{g(X;\theta_t)}\ln g(X;\theta)\Big] \iff \max_{\theta\in\Theta}\mathbf{E}_{\theta_0}\Big[\mathbb{1}_{\{f(y)\leq\gamma_t\}}(X)\ln g(X;\theta)\Big] \tag{10}$$

when θ_t and γ_t is given, since $\mathbf{X}_t^{(1)},\ldots,\mathbf{X}_t^{(N)}$ is *i.i.d* from $g(\cdot;\theta_t)$ and

$$\mathbf{E}_{\theta_0}\Big[\mathbb{1}_{\{f(y)\leq\gamma_t\}}(X)\ln g(X;\theta)\Big] = \mathbf{E}_{\theta'}\Big[\mathbb{1}_{\{f(y)\leq\gamma_t\}}(X)\frac{g(X;\theta_0)}{g(X;\theta')}\ln g(X;\theta)\Big]$$

for any $\theta' \in \Theta$. With γ_t instead of γ in equations (4)-(8), we know that $g(\cdot;\theta_{t+1})$ is an approximation of

$$g_t^*(x) := \frac{\mathbb{1}_{\{f(y)\leq\gamma_t\}}(x)g(x;\theta_0)}{\int_{z\in\Omega}\mathbb{1}_{\{f(y')\leq\gamma_t\}}(z)g(z;\theta_0)dz} \quad \text{for all } x\in\Omega$$

whenever γ_t is fixed. Therefore, if we can make γ_t gradually approximate γ, then we can finally reach an optimal $\theta^* \in \Theta$. As a reference, we write this procedure in Figure 1 on p. 154. The procedure can be seen as an application of the CE algorithm on p. 19 to the field RES. It differs with its counterpart in CO in the update of model. For the case of RES, we use (9), since we aim to estimate $\mathbb{P}(\mathfrak{A})$ and (10) shows an equivalence. However, for the case of CO, we estimate an empirical model $\mathbf{W}_t = g(\cdot;\theta_{t+1})$ by

$$\max_{\theta\in\Theta} \frac{1}{N}\sum_{j=1}^{N} \mathbb{1}_{\{f(y)\leq\gamma_t\}}(\mathbf{X}^{(j)})\ln g(\mathbf{X}_t^{(j)};\theta).$$

and then we construct the next model $\mathbf{\Pi}_{t+1}$ as a convex combination of the present model $\mathbf{\Pi}_t$ and $\mathbf{W}_{t+1}$, where we identify Θ as $\mathbb{P}_{ce}$ and $g(\cdot;\theta)$ as θ for each $\theta \in \mathbb{P}_{ce}$. For a more detailed explanation, see [Rub99].

Let $\{\gamma_t^*\}_{t\in\mathbb{N}}$ and $\{\theta_t^*\}_{t\in\mathbb{N}}$ be two sequences satisfying

- $\theta_0^* = \theta_0 \in \Theta$,
- γ_t^* is the theoretical value such that $\mathbf{E}_{\theta_t^*}\big[\mathbb{1}_{\{f(y)\leq\gamma_t^*\}}(X)\big] = \alpha$,
- $\theta_{t+1}^* \in \Theta$ is the theoretical solution to

$$\max_{\theta\in\Theta}\mathbf{E}_{\theta_0}\Big[\mathbb{1}_{\{f(y)\leq\gamma_t^*\}}(X)\ln g(X;\theta)\Big],$$

for $t = 0, 1, \ldots$. Obviously, γ_t and θ_t are estimators of γ_t^* and θ_t^* respectively for each $t \in \mathbb{N}$. Let $\Gamma : \Theta \mapsto \mathbb{R}$ be a map such that $\Gamma(\theta)$ is the theoretical value such that

$$\mathbf{E}_\theta\big[\mathbb{1}_{\{f(y)\leq\Gamma(\theta)\}}(X)\big] = \alpha$$

for each $\theta \in \Theta$. Then $\gamma_t^* = \Gamma(\theta_t^*)$ for each $t \in \mathbb{N}$. D. Lieber shows in his Thesis [Lie98] that

Lemma 8.1 (see also Theorem 1.1 in [Rub99]). *Assume that for each real number $c \in \mathbb{R}$, there is a unique solution to*

$$max_{\theta\in\Theta}\mathbf{E}_{\theta_0}\Big[\mathbb{1}_{\{f(y)\leq c\}}(X)lng(X;\theta)\Big],$$

and we use $\eta(c) \in \Theta$ to denote the unique solution ($\theta^*_{t+1} = \eta(\gamma^*_t)$). And suppose further that

- *$\{\gamma^*_t\}_{t\in\mathbb{N}}$ is monotonically increasing,*
- *the map $\Gamma : \Theta \mapsto \mathbb{R}$ is continuous,*
- *the map $\eta : \mathbb{R} \mapsto \Theta$ is proper, i.e. if c belongs to an interval, then $\eta(c)$ belongs to a compact set,*
- *the map $\Gamma\Big(\eta\big(\Gamma(\theta)\big)\Big) - \Gamma(\theta)$ is semi-continuous.*

Then

$$\lim_{N\to\infty}\mathbf{P}\big[\gamma_0 > \gamma_1 > \gamma_2 > \cdots > \gamma_t > \cdots\big] = 1 \text{ and } \mathbf{P}\big[\exists t \in \mathbb{N}, \gamma_t \leq \gamma\big] = 1.$$

Proof. see [Lie98]. □

Lemma 8.1 actually shows that under some 'mild condition', the procedure in Figure 1 may stop with an optimal $\theta^* \in \Theta$ in finite many iterations if a suitable sample size is employed.

Algorithm Cross entropy algorithm for rare event simulation

parameters:

parametric family $\mathbb{P}_{IS} = \{g(\cdot;\theta)|\theta \in \Theta\}$, an $\alpha \in (0,1)$ not very small, a sample size N.

algorithm:

a) Generate random sample $\mathbf{X}_t^{(1)}, \ldots, \mathbf{X}_t^{(N)}$ $i.i.d$ with $g(\cdot;\theta_t)$, and order them according to f as $f(\mathbf{X}_t^{(m_1)}) \leq \cdots \leq f(\mathbf{X}_{(m_N)})$, set $\gamma_t = f(\mathbf{X}_t^{(m_{\lfloor \alpha \cdot N \rfloor})})$;

b) Solve

$$\max_{\theta \in \Theta} \frac{1}{N} \sum_{j=1}^{N} \mathbb{1}_{\{f(y) \leq \gamma_t\}}(\mathbf{X}^{(j)}) \frac{g(\mathbf{X}_t^{(j)};\theta_0)}{g(\mathbf{X}_t^{(j)};\theta_t)} \ln g(\mathbf{X}_t^{(j)};\theta)$$

and denote the solution as θ_{t+1};

c) while $(\gamma_t > \gamma)$

c 1) $t = t + 1$;

c 2) repeat step **a)**-**c)**;

d) output θ_t;

Figure 1: Cross entropy algorithm for rare event simulation

A.7 Genetic algorithms, simulated annealing and particle swarm optimization

A.7.1 Genetic algorithms

GAs are inspired by the natural evolution of genes of creatures. For a tutorial and overview, see [Mic96] and [Whi94].

In GAs, problems are generally unconstrained 0-1 encoded. Therefore, we may assume an instance (S, f) here with $S = \{0,1\}^L$ for an $L \in \mathbb{N}$.

Different from MBS, in GAs we call a collection of feasible solutions as a *population*. The input parameters for GAs are: a fixed *parent population size* $N \in \mathbb{N}$, a fixed *offspring population size* $M \in \mathbb{N}$, a constant *mutation rate* $r_m \in (0,1)$. The initial parent population P_0 are uniformly sampled from S. And, in each sequent iteration t, a fixed *selection strategy* is employed to choose $M/2$ pairs of solutions from P_t. After that, a *crossover operator* applies on each pair to produce two offspring, and each offspring may encounter a *mutation* with probability r_m. Here, we employ O_t to denote the resulted collection of offspring. Then, N many solutions in $O_t \cup P_t$ would be selected to form P_{t+1} by a fixed *population update strategy*.

In GAs, typical selection strategies are the *roulette selection* and *tournament selection*. In a roulette selection, we randomly pick out M many solutions from P_t by probabilities proportional to qualities of solutions. Then, the first pair is formed by the first and second selected solutions, the second pair is formed by the third and fourth selected solutions and so on. Tournament selection is similar with a real tournament competition with a fixed group size.

A crossover operator can be mathematically formulated as a random function which takes in two parents and outputs two children. Here, we take 1-point crossover as an example. Given two solutions $s^{(1)} = (s_1^{(1)}, s_2^{(1)}, \ldots, s_L^{(1)})$ and $s^{(2)} = (s_1^{(2)}, s_2^{(2)}, \ldots, s_L^{(2)})$, and a randomly picked crossover point j from $\{1, \ldots, L\}$. Then 1-point crossover produces two offspring as

$$o^{(1)} = (s_1^{(1)}, \ldots, s_j^{(1)}, s_{j+1}^{(2)}, \ldots, s_L^{(2)}) \text{ and } o^{(2)} = (s_1^{(2)}, \ldots, s_j^{(2)}, s_{j+1}^{(1)}, \ldots, s_L^{(1)}).$$

A typical mutation operator in GAs is the 1-flip. Given a solution $(s_1, \ldots, s_L)$ as well as a random flip position i. Then 1-flip will mutate the solution into

$$(s_1, \ldots, s_{i-1}, 1 - s_i, s_{i+1}, \ldots, s_L).$$

A population update strategy generally tells how to form the next parent population from the current parent population and current offspring population. Typical update strategies may be the so-called $\lambda+\mu$ rule and (λ, μ) rule. In $\lambda+\mu$ rule, we will take the best N many solutions in $P_t \cup O_t$ as the next population P_{t+1}. In (λ, μ) rule, P_{t+1} would be a random subsample from O_t with selection probabilities proportional to qualities of qualities.

As a summary, we list the general GA algorithm in Figure 2 on p. 156.

Algorithm Genetic algorithm

a) randomly select N solutions as initial population P_0;

b) set $t = 0$ and a stop criterion $STOP$;

c) select $M/2$ pairs of solutions from P_t;

d) for each pair, do crossover and collect the offspring in O_t;

e) for each solution in O_t, draw a random variate u from $U[0, 1]$, if $u < r_m$, do mutation;

f) build P_{t+1} from $O_t \cup P_t$ with a population update strategy;

g) while $STOP$ does not hold

g 1) $t = t + 1$;

g 2) repeat steps $c) - g)$;

Figure 2: Genetic algorithm

A.7.2 Simulated annealing

Roughly speaking, an LS algorithm starts with a randomly chosen solution, and then iteratively searches a neighborhood for a possible improvement, see [AL97] for an overview of LS in CO. Among those LS algorithms, simulated annealing is the most well-know one.

Simulated annealing generally has two technical components: a *neighborhood structure* and a *temperature decreasing rule*. Given a CO instance (S, f), a neighborhood can be mathematically stated as a function $\mathcal{N} : S \mapsto \{B : B \subseteq S\}$. Neighborhoods are used to restrict the search in each iteration. I.e. in iteration t, a neighbor Y_t would be randomly picked out from the neighborhood $\mathcal{N}(X_t)$ where X_t is the current solution. A temperature decreasing rule is used to update the present temperature. Figure 3 on p. 157 lists the pseudo code for simulated annealing.

Typical examples of neighborhoods are the various kinds of position-exchanges, see [AL97] pp. 4-10 for a reference. In simulated annealing, the present solution is update according to the acceptance/rejection criterion,

$$X_{t+1} := \begin{cases} Y_t & \text{if } f(Y_{t+1}) \leq f(X_t), \\ Y_t & \text{if } f(Y_{t+1}) > f(X_t) \text{ and } u \leq \exp\left(\frac{f(X_t)-f(Y_t)}{T_t}\right), \\ X_t & \text{otherwise,} \end{cases} \tag{11}$$

where T_t is the current temperature, u is a random variate from $U[0, 1]$.

Algorithm Simulated annealing

a) randomly choose an X_0 from S, and fix an initial temperature T_0;

b) set $t = 0$ and a stop criterion $STOP$;

c) randomly choose an Y_t from $\mathcal{N}(X_t)$;

d) choose X_{t+1} from $\{X_t, Y_t\}$ according to (11);

e) update temperature;

f) while $STOP$ does not hold

f1) $t = t + 1$;

f2) repeat steps $c) - f)$;

Figure 3: Simulated annealing

A.7.3 Particle swarm optimization

Particle swarm optimization (PSO, [KE95]) simulates the collective behavior of a school of fishes or birds. It generally assumes a continuous search space S. It employs a fixed number of agents iteratively searching on S. Initially, the agents randomly construct a starting position (solution) on S. In each subsequent iteration, the agents move their present positions towards their own best positions and the global best position. The agents iteratively move their positions as following:

Initialization Randomly generate a swarm of positions of a fixed size N :

$$x_i = (x_{i,1}, \ldots, x_{i,d}) \in S, \quad i = 1, \ldots, N,$$

where d is the dimension. Initialize the N velocity vectors

$$v_i^0 = (v_{i,1}^0, \ldots, v_{i,d}^0) \in \mathbb{R}^d, \quad i = 1, \ldots, N.$$

Let b_i denote the best position where agent i experienced, and b denote the global best position the swarm has ever visited so far. Take an *inertia* parameter e, and *cognitive* parameter c_1 and a *social* parameter c_2 from $[0, 1]$.

Update Local and Global Information If it is the first iteration, then $b_i = x_i$ and b is the best position in the first swarm. Otherwise, if x_i is better than b_i, set $b_i = x_i$. And if b is worse than the best position in the current swarm, set b to be that position.

Update Velocity Generate two random numbers r_1, r_2 uniformly from $[0, 1]$ and update velocities with the following form:

$$v_i^{t+1} = e \cdot v^t + r_1 \cdot c_1 \cdot (b_i - x_i) + r_2 \cdot c_2 \cdot (b - x_i), \tag{12}$$

for each $i = 1, \ldots, N$.

Update Position Each position is updated by

$$x_i = x_i + v_i^{t+1}, \tag{13}$$

for $i = 1, \ldots, N$.

Although PSO is initially designed for continuous optimization, some recent literature have successfully applied it to combinatorial optimization, see e.g. [WHZP03]..

Bibliography

[AB09] Sanjeev Arora and Boaz Barak, *Computational complexity: a modern approach*, Cambridge University Press, 2009.

[AEP05] N. Andréasson, A. Evggrafov, and M. Patriksson, *An introduction to continuous optimization: Foundations and fundamental algorithms*, Studentlitteratur, 2005.

[AKRR05] G Alon, Dirk P Kroese, Tal Raviv, and Reuven Y Rubinstein, *Application of the cross-entropy method to the buffer allocation problem in a simulation-based environment*, Annals of Operations Research **134** (2005), no. 1, 137–151.

[AL97] EH Emile HL Aarts and Jan K Lenstra, *Local search in combinatorial optimization*, Princeton University Press, 1997.

[Alp10] Ethem Alpaydin, *Introduction to machine learning. 2nd*, 2010.

[AM94] Hideki Asoh and Heinz Mühlenbein, *On the mean convergence time of evolutionary algorithms without selection and mutation*, Parallel Problem Solving from NaturePPSN III, Springer, 1994, pp. 88–97.

[Bal94] Shumeet Baluja, *Population-based incremental learning. a method for integrating genetic search based function optimization and competitive learning*, Tech. report, DTIC Document, 1994.

[BF03] Ş İlker Birbil and Shu-Chering Fang, *An electromagnetism-like mechanism for global optimization*, Journal of global optimization **25** (2003), no. 3, 263–282.

[BHS97] Bernd Bullnheimer, Richard F Hartl, and Christine Strauss, *A new rank based version of the ant system. a computational study.*, SFB Adaptive Information Systems and Modelling in Economics and Management Science, WU Vienna University of Economics and Business, 1997.

[BHS99] ———, *Applying the ant system to the vehicle routing problem*, Meta-Heuristics, Springer, 1999, pp. 285–296.

[BR03] Christian Blum and Andrea Roli, *Metaheuristics in combinatorial optimization: Overview and conceptual comparison*, ACM Computing Surveys (CSUR) **35** (2003), no. 3, 268–308.

[BT93] Dimitris Bertsimas and John Tsitsiklis, *Simulated annealing*, Statistical Science (1993), 10–15.

[CDMT94] Alberto Colorni, Marco Dorigo, Vittorio Maniezzo, and Marco Trubian, *Ant system for job-shop scheduling*, Belgian Journal of Operations Research, Statistics and Computer Science **34** (1994), no. 1, 39–53.

[CdVHM00] Oscar Cordon, Iñaki Fernández de Viana, Francisco Herrera, and Llanos Moreno, *A new aco model integrating evolutionary computation concepts: The best-worst ant system*, Citeseer, 2000.

[CHdM05] Krishna Chepuri and Tito Homem-de Mello, *Solving the vehicle routing problem with stochastic demands using the cross-entropy method*, Annals of Operations Research **134** (2005), no. 1, 153–181.

[Cin75] Erhan Cinlar, *Introduction to stochastic processes*, Printice-Hall, Englewood Cliffs, NJ, 1975.

[CJK07] Andre Costa, Owen Dafydd Jones, and Dirk Kroese, *Convergence properties of the cross-entropy method for discrete optimization*, Operations Research Letters **35** (2007), no. 5, 573–580.

[CT12] Thomas M Cover and Joy A Thomas, *Elements of information theory*, John Wiley & Sons, 2012.

[CTCY10] Tianshi Chen, Ke Tang, Guoliang Chen, and Xin Yao, *Analysis of computational time of simple estimation of distribution algorithms*, Evolutionary Computation, IEEE Transactions on **14** (2010), no. 1, 1–22.

[DBIV+97] Jeremy S De Bonet, Ch L Isbell, Paul Viola, et al., *Mimic: Finding optima by estimating probability densities*, Advances in neural information processing systems (1997), 424–430.

[DBKMR05] Pieter Tjerk De Boer, Dirk P Kroese, Shie Mannor, and Reuven Y Rubinstein, *A tutorial on the cross-entropy method*, Annals of operations research **134** (2005), no. 1, 19–67.

[DG97a] Marco Dorigo and Luca Maria Gambardella, *Ant colonies for the travelling salesman problem*, BioSystems **43** (1997), no. 2, 73–81.

[DG97b] ———, *Ant colony system: A cooperative learning approach to the traveling salesman problem*, Evolutionary Computation, IEEE Transactions on **1** (1997), no. 1, 53–66.

[DJ07] Benjamin Doerr and Daniel Johannsen, *Refined runtime analysis of a basic ant colony optimization algorithm.*, IEEE Congress on Evolutionary Computation, 2007, pp. 501–507.

[DMC96] Marco Dorigo, Vittorio Maniezzo, and Alberto Colorni, *Ant system: optimization by a colony of cooperating agents*, Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on **26** (1996), no. 1, 29–41.

[DNSW07] Benjamin Doerr, Frank Neumann, Dirk Sudholt, and Carsten Witt, *On the runtime analysis of the 1-ant aco algorithm*, Proceedings of the 9th annual conference on Genetic and evolutionary computation, ACM, 2007, pp. 33–40.

[DS04] Marco Dorigo and Thomas Stützle, *Ant colony optimization*, Cambridge, Massachusetts: A Bradford Book, MIT Press, 2004.

[DS10] ———, *Ant colony optimization: overview and recent advances*, Handbook of metaheuristics, Springer, 2010, pp. 227–263.

[Dur10] Rick Durrett, *Probability: theory and examples*, vol. 3, Cambridge university press, 2010.

[GD+95] Luca Maria Gambardella, Marco Dorigo, et al., *Ant-q: A reinforcement learning approach to the traveling salesman problem*, ICML, 1995, pp. 252–260.

[GL95] M Grötschel and L Lovász, *Combinatorial optimization*, Handbook of combinatorics **2** (1995), 1541–1597.

[GL99] Fred Glover and Manuel Laguna, *Tabu search*, Springer, 1999.

[GM02] Michael Guntsch and Martin Middendorf, *A population based approach for aco*, Applications of Evolutionary Computing, Springer, 2002, pp. 72–81.

[Gou06] Nicholas Gould, *An introduction to algorithms for continuous optimization*, Oxford University Computing Laboratory Notes, 2006.

[Gut00] Walter J Gutjahr, *A graph-based ant system and its convergence*, Future Generation Computer Systems **16** (2000), no. 8, 873–888.

[Gut02] ———, *Aco algorithms with guaranteed convergence to the optimal solution*, Information processing letters **82** (2002), no. 3, 145–153.

[Gut03] ———, *A generalized convergence result for the graph-based ant system metaheuristic*, Probability in the Engineering and Informational Sciences **17** (2003), no. 4, 545–569.

[Gut07] ———, *Mathematical runtime analysis of aco algorithms: Survey on an emerging issue*, Swarm Intelligence **1** (2007), no. 1, 59–79.

[Gut08] ———, *First steps to the runtime complexity analysis of ant colony optimization*, Computers & Operations Research **35** (2008), no. 9, 2711–2727.

[HGC95] David Heckerman, Dan Geiger, and David M Chickering, *Learning bayesian networks: The combination of knowledge and statistical data*, Machine learning **20** (1995), no. 3, 197–243.

[HLG99] Georges R Harik, Fernando G Lobo, and David E Goldberg, *The compact genetic algorithm*, Evolutionary Computation, IEEE Transactions on **3** (1999), no. 4, 287–297.

[HP11] Mark Hauschild and Martin Pelikan, *An introduction and survey of estimation of distribution algorithms*, Swarm and Evolutionary Computation **1** (2011), no. 3, 111–128.

[Jen06] Johan Ludwig William Valdemar Jensen, *Sur les fonctions convexes et les inégalités entre les valeurs moyennes*, Acta Mathematica **30** (1906), no. 1, 175–193.

[KE95] J Kennedy and R Eberhart, *Particle swarm optimization*, Neural Networks, 1995. Proceedings., IEEE International Conference on, vol. 4, IEEE, 1995, pp. 1942–1948.

[KGOK12] Dervis Karaboga, Beyza Gorkemli, Celal Ozturk, and Nurhan Karaboga, *A comprehensive survey: artificial bee colony (abc) algorithm and applications*, Artificial Intelligence Review (2012), 1–37.

[Kno90] K. Knopp, *Theory and application of infinite series*, Courier Dover Publisher, 1990.

[KV02] Bernhard Korte and Jens Vygen, *Combinatorial optimization: theory and algorithms*, second ed., Springer, 2002.

[Law01] Eugene L Lawler, *Combinatorial optimization: networks and matroids*, DoverPublications. com, 2001.

[LDM09] Manuel Laguna, Abraham Duarte, and Rafael Martí1, *Hybridizing the cross-entropy method: An application to the max-cut problem*, Computers & Operations Research **36** (2009), no. 2, 487–498.

[Lie98] D. Lieber, *Rare events estimation via cross entropy and importance sampling*, William Davidson Faculty of Industrial Engineering and Management, Technion, Haifa, Israel, 1998.

[Mar05] Leonid Margolin, *On the convergence of the cross-entropy method*, Annals of Operations Research **134** (2005), no. 1, 201–214.

[Mic96] Zbigniew Michalewicz, *Genetic algorithms+ data structures= evolution programs*, springer, 1996.

[ML09] Tai-Yu Ma and Jean-Patrick Lebacque, *A cross entropy based multi-agent approach to traffic assignment problems*, Traffic and Granular Flow07, Springer, 2009, pp. 161–170.

[MMR99] Heinz Mühlenbein, Thilo Mahnig, and Alberto Ochoa Rodriguez, *Schemata, distributions and graphical models in evolutionary optimization*, Journal of Heuristics **5** (1999), no. 2, 215–247.

[MP96] H Mühlenbein and G Paaß, *From recombination of genes to the estimation of distributions i. binary parameters*, Parallel Problem Solving from NaturePPSN IV, Springer, 1996, pp. 178–187.

[Nic07] TAJ Nicholson, *Optimization in industry: Optimization techniques*, vol. 1, Transaction Publishers, 2007.

[NW06] Frank Neumann and Carsten Witt, *Runtime analysis of a simple ant colony optimization algorithm*, Springer, 2006.

[NW09] _____, *Runtime analysis of a simple ant colony optimization algorithm*, Algorithmica **54** (2009), no. 2, 243–255.

[OHS+11] S. M. Oliveira, M. S. Hussin, T. Stuetzle, A. Roli, and M. Dorigo, *A detailed analysis of the population-based ant colony optimization algorithm for the tsp and the qap*, GECCO'11 Proceedings of the 13th annual conference companion on the Genetic and evolutionary computation (New York, NY, USA), 2011, pp. 13–14.

[Pel05] Martin Pelikan, *Bayesian optimization algorithm*, Hierarchical Bayesian Optimization Algorithm, Springer, 2005, pp. 31–48.

[PGCP00] Martin Pelikan, David E Goldberg, and Erick Cantu-Paz, *Linkage problem, distribution estimation, and bayesian networks*, Evolutionary computation **8** (2000), no. 3, 311–340.

[PGL02] Martin Pelikan, David E Goldberg, and Fernando G Lobo, *A survey of optimization by building and using probabilistic models*, Computational optimization and applications **21** (2002), no. 1, 5–20.

[PM98] Martin Pelikan and Heinz Mühlenbein, *Marginal distributions in evolutionary algorithms*, Proceedings of the International Conference on Genetic Algorithms Mendel, vol. 98, Citeseer, 1998, pp. 90–95.

[RK04] Reuven Y Rubinstein and Dirk P Kroese, *The cross-entropy method: a unified approach to combinatorial optimization, monte-carlo simulation and machine learning*, Springer, 2004.

[RP95] Motwani Rajeevi and Raghavan Prabhakar, *Randomized algorithms*, Cambridge University Press, 1995.

[RT+09] Gerardo Rubino, Bruno Tuffin, et al., *Rare event simulation using monte carlo methods*, Wiley Online Library, 2009.

[Rub97] Reuven Y Rubinstein, *Optimization of computer simulation models with rare events*, European Journal of Operational Research **99** (1997), no. 1, 89–112.

[Rub99] ———, *The cross-entropy method for combinatorial and continuous optimization*, Methodology and computing in applied probability **1** (1999), no. 2, 127–190.

[Rub02] ———, *Cross-entropy and rare events for maximal cut and partition problems*, ACM Transactions on Modeling and Computer Simulation (TOMACS) **12** (2002), no. 1, 27–53.

[SBW11] Budi Santosa, Muhammad Arif Budiman, and Stefanus Eko Wiratno, *A cross entropy-genetic algorithm for m-machines no-wait job-shop scheduling problem*, Journal of Intelligent Learning Systems and Applications **3** (2011), 171–180.

[SG00] Kumara Sastry and David E Goldberg, *On extended compact genetic algorithm*, Late-Breaking Paper at the Genetic and Evolutionary Computation Conference, 2000, pp. 352–359.

[SH00] Thomas Stützle and Holger H Hoos, *Max–min ant system*, Future generation computer systems **16** (2000), no. 8, 889–914.

[SVVdW80] Edward A Silver, R Victor, V Vidal, and Dominique de Werra, *A tutorial on heuristic methods*, European Journal of Operational Research **5** (1980), no. 3, 153–162.

[Tra84] Boris A Trakhtenbrot, *A survey of russian approaches to perebor (brute-force searches) algorithms*, Annals of the History of Computing **6** (1984), no. 4, 384–400.

[Whi94] Darrell Whitley, *A genetic algorithm tutorial*, Statistics and computing **4** (1994), no. 2, 65–85.

[WHZP03] Kang-Ping Wang, Lan Huang, Chun-Guang Zhou, and Wei Pang, *Particle swarm optimization for traveling salesman problem*, Machine Learning and Cybernetics, 2003 International Conference on, vol. 3, IEEE, 2003, pp. 1583–1585.

[WK14a] Zijun Wu and Michael Kolonko, *Absorption in model-based search algorithms for combinatorial optimization*, Evolutionary Computation (CEC), 2014 IEEE Congress on, IEEE, 2014, pp. 1744–1751.

[WK14b] ———, *Asymptotic properties of a generalized cross entropy optimization algorithm*, Evolutionary Computation, IEEE Transactions on **18** (2014), no. 5, 658–673.

[WM97] David H Wolpert and William G Macready, *No free lunch theorems for optimization*, Evolutionary Computation, IEEE Transactions on **1** (1997), no. 1, 67–82.

[Woe03] Gerhard J Woeginger, *Exact algorithms for np-hard problems: A survey*, Combinatorial OptimizationEureka, You Shrink!, Springer, 2003, pp. 185–207.

[WS11] David P Williamson and David B Shmoys, *The design of approximation algorithms*, Cambridge University Press, 2011.

[ZBMD04] Mark Zlochin, Mauro Birattari, Nicolas Meuleau, and Marco Dorigo, *Model-based search for combinatorial optimization: A critical survey*, Annals of Operations Research **131** (2004), no. 1-4, 373–395.

[ZM04] Qingfu Zhang and Heinz Muhlenbein, *On the convergence of a class of estimation of distribution algorithms*, Evolutionary Computation, IEEE Transactions on **8** (2004), no. 2, 127–136.

Index