A COMPARATIVE STUDY OF METAHEURISTIC ALGORITHMS FOR THE FERTILIZER OPTIMIZATION PROBLEM

A Thesis Submitted to the
College of Graduate Studies and Research
in Partial Fulfillment of the Requirements
for the degree of Master of Science
in the Department of Computer Science
University of Saskatchewan
Saskatoon

By
Dai Chen

PERMISSION TO USE

In presenting this thesis in partial fulfilment of the requirements for a Postgraduate degree from the University of Saskatchewan, I agree that the Libraries of this University may make it freely available for inspection. I further agree that permission for copying of this thesis in any manner, in whole or in part, for scholarly purposes may be granted by the professor or professors who supervised my thesis work or, in their absence, by the Head of the Department or the Dean of the College in which my thesis work was done. It is understood that any copying or publication or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of Saskatchewan in any scholarly use which may be made of any material in my thesis.

Requests for permission to copy or to make other use of material in this thesis in whole or part should be addressed to:

Head of the Department of Computer Science
176 Thorvaldson Building
110 Science Place
University of Saskatchewan
Saskatoon, Saskatchewan
Canada
S7N 5C9

ABSTRACT

Hard combinatorial optimization (CO) problems pose challenges to traditional algorithmic solutions. The search space usually contains a large number of local optimal points and the computational cost to reach a global optimum may be too high for practical use. In this work, we conduct a comparative study of several state-of-the-art metaheuristic algorithms for hard CO problems solving. Our study is motivated by an industrial application called the Fertilizer Blends Optimization. We focus our study on a number of local search metaheuristics and analyze their performance in terms of both runtime efficiency and solution quality. We show that local search granularity (move step size) and the downhill move probability are two major factors that affect algorithm performance, and we demonstrate how experimental tuning work can be applied to obtain good performance of the algorithms.

Our empirical result suggests that the well-known Simulated Annealing (SA) algorithm showed the best performance on the fertilizer problem. The simple Iterated Improvement Algorithm (IIA) also performed surprisingly well by combining strict uphill move and random neighborhood selection. A novel approach, called Delivery Network Model (DNM) algorithm, was also shown to be competitive, but it has the disadvantage of being very sensitive to local search granularity. The constructive local search method ($GRASP_{IIA}$), which combines heuristic space sampling and local search, outperformed IIA without a construction phase; however, the improvement in performance is limited and generally speaking, local search performance is not sensitive to initial search positions in our studied fertilizer problem.

ACKNOWLEDGEMENTS

First, I would sincerely like to thank Dr. Michael Horsch, my supervisor, for his guidance, support, and constant encouragement throughout my two years' research. Mike's inspiration has been an important factor for my accomplishment of the thesis work. I would also like to thank Western Ag Inc. and PRECARN Corp. for funding this research project. Specific thanks go to my program committee members, Dr. Nathaniel Osgood, Dr. Raymond Spiteri, and Dr. Murray Bremner (external), for their suggestions and insights. The Department of Computer Science at the University of Saskatchewan has been such an encouraging place that I am proud of being one member of it. Also thanks go to many of my colleagues in the department, especially people in the ARIES research lab, for their help and support.

DEDICATED TO MY PARENTS

XUE CHAO DAI AND YONG FEN CHEN

CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ACO	Ant Colony Optimization
AP	Algorithm Performance
ATSP	Asymmetric Traveling Salesman Problem
BDSP	Bus Driver Scheduling Problem
BPP	Bin Packing Problem
CL	Candidate List
CO	Combinatorial Optimization
CSP	Constraint Satisfaction Problem
DNM	Delivery Network Model
ETSP	Euclidean Traveling Salesman Problem
GRASP	Greedy Randomized Adaptive Search Procedure
GRASP$_{IIA}$	GRASP using Iterative Improvement Algorithm as the local search phase
GRASP$_{SA}$	GRASP using Simulated Annealing as the local search phase
GRASP$_{DNM}$	GRASP using DNM algorithm as the local search phase
IIA	Iterative Improvement Algorithm
LS	Local Search
MAGMA	MultiAGent Metaheuristic Architecture
MCOP	Multi-objective Combinatorial Optimization Problem
MDVRP	Multiple Depot Vehicle Routing Problem
MPS	Meta-Parameter Setting
NRO	Network Routing Optimization
OR	Operations Research
PSO	Particle Swarm Optimization
QAP	Quadratic Assignment Problem
QR	Quality Runtime measurement
RCL	Restricted Candidate List
RQ	Runtime Quality measurement
RTD	Run Time Distribution measurement
SA	Simulated Annealing
SLS	Stochastic Local Search
SI	Swarm Intelligence
TSP	Traveling Salesman Problem
TS	Tabu Search
VNDS	Variable Neighborhood Decomposition Search
VNS	Variable Neighborhood Search
VRPTW	Vehicle Routing Problem with Time Windows
VRP	Vehicle Routing Problem

LIST OF SYMBOLS

$\mathbf{A}$	The nutrients mixture table
$\mathbf{B}$	The compound blends
$\mathbf{d}_i$	The blend delivered at site i
$\mathbf{g}_i$	Existing nutrients delivered at site i
$\mathbf{m}_i$	The ith market nutrient
$\mathbf{M}$	The market nutrients set
$\mathbf{o}_i$	Maximum allowable nutrients at site i
$\mathbf{S}$	The solution space of an optimization problem
$\mathbf{V}$	Delivery rate table
$\mathbf{v}_j$	The delivery rate vector at site j
α	Micro tuning step size in DNM local search
Δa	The local search step size for changing $\mathbf{A}$ table in SA and IIA
δ	Augmentation step size in DNM local search
ΔE	Change in fitness function value $f(s') - f(s)$
Δv	The local search step size for changing $\mathbf{V}$ table in SA and IIA
e	Solution element in GRASP
η	The heuristic value associated with the market nutrient in DNM local search
$f(s)$	The fitness function of an optimization problem
$g(e)$	The greedy function in GRASP
$\hat{a}$	Attractor of the basin
$\hat{s}$	Local optimum solution point
k_B	Boltzmann constant in Simulated Annealing
k	Total number of farm sites
$N'(s)$	The allow set in Tabu Search
$N(s)$	The neighborhood of s
n	Total number of market nutrients
p_i	The net profit at site i
$\pounds$	Fitness landscape
P	The total net profit in the fertilizer problem
α	RCL selection parameter in GRASP
s^*	Global optimum solution point
T	Temperature level in Simulated Annealing
φ	Transient operator
ρ	Solution quality threshold
ϵ	Neighborhood granularity, or landscape granularity
Y_i	Crop yield at site i

CHAPTER 1

INTRODUCTION

Optimization refers to problems that require searching for a best configuration of a set of variables to achieve certain goal. In optimization, there is a class of problems called Combinatorial Optimization (CO) Problems [5, 55]. In CO, solutions are encoded with discrete variables, and the search process usually involves exploring a problem space represented by sets of values, matrices, or graphs. Especially in recent years, work in CO has been motivated by its various applications in both academic and industrial domains.

Hard combinatorial optimization problems usually feature a high number of degrees of freedom, non-linearity, and the existence of large number of local optima [37, 20, 7]. The intrinsic difficulty of these problems often leads to intractable amount of computational time for solving them. As a target problem and case study, we consider the Fertilizer Blends Optimization Problem. The fertilizer optimization problem deals with nutrient delivery during planting. The goal is to optimize the total net profit in the field using an optimal nutrient delivery scheme. The delivery scheme defines what nutrients should be delivered to each farm site, as well as the quantity applied. Because nutrient delivery affects both the site output (the yield) and delivery cost, solving the problem of optimizing the total net profit requires searching among all possible nutrient delivery schemes to obtain the best one.

The difficulty of the problem comes from the fact that each field on a farm contains many sites, and each site has its specific need for a particular kind of nutrient blend. If we want to optimize the yield of one site, we could deliver exactly the required nutrient blend for it. However, this method becomes impractical for a large number of sites: both the work effort and financial cost is too high for obtaining different nutrient blends for different sites. One feasible way is to find a compound blend that is a mixture of several available market nutrients. We then deliver the blend to different sites at different rates. Because current seeding technology allows two independent blends to be applied at the same time, two compound blends can be used. The solution for a given fertilizer problem instance (e.g., a specific farm field) contains two tables: a “Mixture” table $\mathbf{A}$ and a “Delivery Rate” table $\mathbf{V}$. Table $\mathbf{A}$ defines how several existing market nutrients are mixed together to produce the two compound blends; table $\mathbf{V}$, on the other hand, specifies the quantity of the blend applied at each site.

Because one farm field may contain hundreds of sites, a systematic search method is infeasible. The algorithm not only searches among all possible nutrient combinations for the compound blend, but also the possible delivery rate combinations for all sites. Finding an optimal solution involves exploring the combined high-dimensional space of both the “Mixture” table and “Delivery Rate” table.

Hard combinatorial optimization problems pose challenges to search algorithms. The solution space usually contains a large number of local optimal points, and the computational cost for reaching a global optimum may be too high for practical use. In this work, we apply metaheuristic algorithms to solve the fertilizer problem. Metaheuristic algorithms [27, 37, 5] belong to the class of approximation search algorithms. They differ from traditional heuristic search [66] in that a set of high level strategies are usually integrated into the search framework in order to explore the solution space both effectively and efficiently.

In this thesis, we conduct a comparative study of several state-of-the-art metaheuristic algorithms for the fertilizer optimization problem. We focus our study on a number of local search metaheuristics and analyze their performance in terms of both runtime efficiency and solution quality.

Simulated Annealing (SA) [35, 46] is one of the well-established local search metaheuristics that has explicit strategies to escape from local optima. The idea originates from how a low energy atom configuration is found in statistical mechanics [46]. In SA, “temperature” is used as a meta-parameter to help direct the search. A cooling schedule specifies how the temperature level drops adaptively from high to low. The basic idea is to keep the algorithm actively exploring the solution space at the very beginning of the search but force it to focus on refining the solutions found, so that high quality solutions are more likely to be obtained.

We also propose in this work a delivery flow network model [3] for the fertilizer problem. The idea is to build the solution incrementally by considering the nutrient delivery process as a resource-allocation problem [55]. We transform the problem of searching for an optimal delivery scheme into the problem of finding a special nutrient flow in the network that optimizes the network utility. Utility here refers to total net profit. A Delivery Network Model (DNM) local search algorithm is then presented to solve the problem. Our empirical results show that the DNM local search algorithm is an efficient algorithmic candidate for the fertilizer problem. However, DNM local search performance is very sensitive to its meta-parameter settings, and considerable experimental effort is required for performance tuning.

An alternative approach to local search metaheuristics (SA and DNM) is to use a construction method with local search. In this work, we apply the Greedy Randomized Adaptive Search Procedure (GRASP) [61, 34, 37] to the fertilizer problem. There are two major phases in GRASP: solution construction followed by local search. In the first phase, we construct the solution incrementally using a greedy randomized construction procedure. Once a complete solution is obtained, we apply the local search to perform an intensified regional search. Existing literature [61, 34] reports that combining a construction method with local search usually yields higher solution quality than local search alone.

The results of this work are several algorithmic candidates that solve the fertilizer optimization problem in terms of acceptable levels of runtime efficiency and solution quality. Furthermore, we show that algorithms performance are affected by their meta-parameter settings, and demonstrate how experimental tuning work could be applied to obtain a good performance of our studied algorithms. We also investigate in this work how the balance between search diversification and intensification is maintained during the search process, and study how these factors affect the overall performance of the various metaheuristic algorithms.

The remainder of the thesis is structured as follows: in Chapter 2, we give a formal definition of the fertilizer blends optimization problem. In Chapter 3, we provide a brief introduction to the general field of combinatorial optimization. We emphasize the notion of *fitness landscape* and present an abstract view for understanding the search behavior of metaheuristic algorithms. The literature review of some major metaheuristics is given in Chapter 4. Problem-specific application issues are described in Chapter 5. Empirical results, including meta-parameter tuning for each of the addressed algorithms, and the comparison among them, are presented in Chapter 6. Conclusions and future work are given in Chapter 7.

CHAPTER 2

THE FERTILIZER BLENDS OPTIMIZATION PROBLEM

Our study in this work is motivated by an industrial application called Fertilizer Blends Optimization. In the fertilizer blend problem, nutrient blends are delivered to a number of farm sites. There exists two stages in nutrient delivery: the nutrient mixing process and the blend delivery process.

2.1 Nutrient Mixing Process

During the first stage, we combine several existing market nutrients to produce the compound blends. Current farm machinery allows two compound blends to be delivered at the same time as a way of improving the nutrient delivery quality. Thus, the nutrient mixing process produces two blends. The market nutrient is defined as a quantity vector of four chemical compounds[1]

$$\mathbf{m} = \begin{pmatrix} N \\ P \\ K \\ S \end{pmatrix}_{4\times 1}$$

The set of all n market nutrients[2] is

$$\mathbf{M} = \begin{pmatrix} \mathbf{m}_1 & \mathbf{m}_2 & \dots & \mathbf{m}_n \end{pmatrix}_{4\times n}$$

---

[1] The four chemical compounds are *Nitrogen (N), Phosphorus (P), Potassium (K) and Sulfur (S).*

[2] Unless specified explicitly in the paper, n refers to the total number of market nutrients.

To obtain the two compound blends, we mix all market nutrients together using the mixture table $\mathbf{A}$. Table $\mathbf{A}$ defines the fraction of each nutrient $\mathbf{m}$ used in each of the two compound blends, thus having the form

$$\mathbf{A} = \begin{pmatrix} \mathbf{a}_1 & \mathbf{a}_2 \end{pmatrix}_{n\times 2}, \ \mathbf{a}_l = \begin{pmatrix} a_{1l} \\ a_{2l} \\ \vdots \\ a_{nl} \end{pmatrix}_{n\times 1}, \ l = 1, 2$$

where

$$\sum_{i=1}^{n} \{a_{il}\} = 1, \ l = 1, 2.$$

The two compound blends produced are

$$\mathbf{B} = \begin{pmatrix} \mathbf{b}_1 & \mathbf{b}_2 \end{pmatrix}_{4\times 2}, \ \mathbf{b}_l = \begin{pmatrix} N_l \\ P_l \\ K_l \\ S_l \end{pmatrix}_{4\times 1}, \ l = 1, 2.$$

The nutrient mixture process can be formalized as

$$\mathbf{B} = \mathbf{MA}$$

2.2 Blends Delivery Process

In the second stage, we deliver the compound blends $\mathbf{B}$ to various sites in the farm field at different rates. The delivery rate table $\mathbf{V}$ is defined as

$$\mathbf{V} = \begin{pmatrix} \mathbf{v}_1 & \mathbf{v}_2 & \dots & \mathbf{v}_k \end{pmatrix}_{2\times k}$$

where

$$\mathbf{v}_j = \begin{pmatrix} v_{1j} \\ v_{2j} \end{pmatrix}_{2\times 1}, \ j = 1, 2, \dots, k.$$

Here, k is the total number of sites; $\mathbf{v}_j$ represents the delivery rate of applying $\mathbf{B}$ at site j, in lbs per acre. Since two independent blends are delivered at the same time, each $\mathbf{v}_j$ contains two quantity values. The total blend delivered at site j is denoted as

$$\mathbf{d}_j = \mathbf{B}\mathbf{v}_j,\ j = 1, 2, \ldots, k.$$

To represent $\mathbf{d_j}$ in a chemical compounds vector, we have

$$\mathbf{d}_j = \begin{pmatrix} N_j \\ P_j \\ K_j \\ S_j \end{pmatrix}_{4\times 1}$$

The blend delivery process can be formalized as

$$\mathbf{D} = \mathbf{B}\mathbf{V} = \begin{pmatrix} \mathbf{d}_1 & \mathbf{d}_2 & \ldots & \mathbf{d}_k \end{pmatrix}_{4\times k}$$

The whole nutrient delivery process is shown in Figure 2.1.

2.3 Optimizing the Objective Function

The objective function to be optimized is the total net profit. For each site i, there is a known ground nutrient vector representing the amount of nutrients already in the site. This is denoted as

$$\mathbf{g}_i = \begin{pmatrix} N_i^o \\ P_i^o \\ K_i^o \\ S_i^o \end{pmatrix}_{4\times 1}$$

The maximum allowable quantity of nutrients at the site is also known in advance

$$\mathbf{o}_i = \begin{pmatrix} \bar{N}_i \\ \bar{P}_i \\ \bar{K}_i \\ \bar{S}_i \end{pmatrix}_{4\times 1}$$

The crop yield at site i is denoted by Y_i, in bushels per acre. Given the blend delivered at site i as $\mathbf{d}_i$, we are able to estimate the crop yield at that site using the following empirical yield function

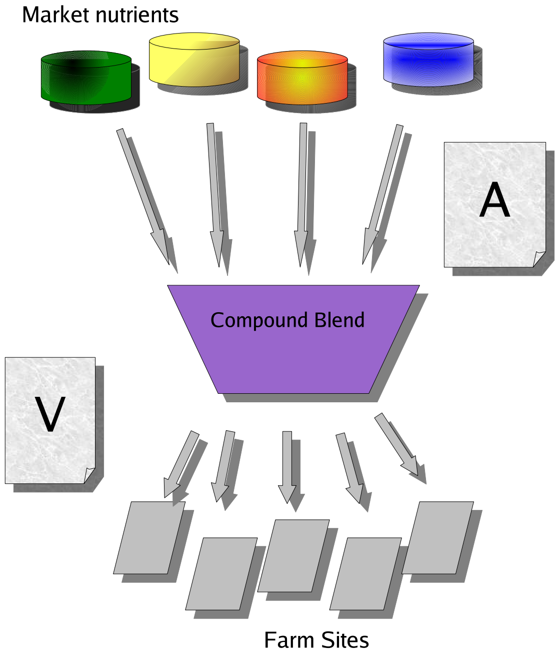

Figure 2.1: Nutrient delivery process. **A** is the nutrient mixture table; **V** is the delivery rate table. We first mix existing market nutrients to produce the compound blend (containing two blends), then deliver the blends to different sites at different rates.

$$\begin{aligned} Y_i &= (\mathbf{d}_i, \mathbf{g}_i, \mathbf{o}_i) \\ &= Y_{\max} f_N\left(\tfrac{N_i+N_i^o}{N_i^*}\right) f_P\left(\tfrac{P_i+P_i^o}{P_i^*}\right) f_K\left(\tfrac{K_i+K_i^o}{K_i^*}\right) f_S\left(\tfrac{S_i+S_i^o}{S_i^*}\right) Z_i \end{aligned}$$

Here, $Y_{\max}$ refers to the maximum yield of the crop and Z_i represents the environmental factors such as temperature and water level; f_N, f_P, f_K, f_S are called "fractional sufficiency" functions that map a given chemical compound quantity to a sufficiency value in the range $[0, 1]$. Both $\mathbf{g}_i$ and $\mathbf{o}_i$ are provided in the site data through an analysis of the farm field; N_i, P_i, K_i, S_i are the four chemical compounds in $\mathbf{d}_i$.

The yield function Y_i is specific to any given farm site and is an approximation of a simulation process which is outside the scope of our work. A detailed analysis of the yield function may lead to analytical methods for solving the problem. However, for the work addressed in this paper, the yield function is generally considered a black-box. Thus the fertilizer problem is a "black-box optimization" problem. On the other hand, our study also shows that investigating the general features[3] of the yield function using analytical methods might provide useful intuitions for the design of problem-specific algorithmic solutions.

The unit market value for the crop is specified by L, in dollars per bushel. Thus, the total market value of a given yield Y_i at site i is denoted by LY_i, in dollars per acre. Cost is also associated with each market nutrient. Given the available market nutrients set $\mathbf{M}$, there is an associated cost vector, denoted as

$$\mathbf{c} = \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{pmatrix}_{n \times 1}$$

Each $c \in \mathbf{c}$ gives the unit cost of using the corresponding market nutrient $\mathbf{m}$ to produce the compound blend, in dollars per unit mass. The total delivery cost at site i is

$$T_i = \mathbf{c}^T \mathbf{A} \mathbf{v}_i$$

in dollars per acre. Here $\mathbf{c}^T$ refers to the matrix transpose of $\mathbf{c}$. The net profit at site i, considering the nutrient delivery cost, is

$$p_i = (LY_i - T_i)R_i$$

in dollars. Here, R_i is the site area in acres. The total net profit is computed by summing together the net profit at all sites, thus having the form

$$P = \sum_{i=1}^{k} p_i = \sum_{i=1}^{k} (LY_i - T_i)R_i$$

---

[3] For example, the factional sufficiency curves f_N, f_P, f_K, f_S; the nutrient overdose effect, and the commonality of $\mathbf{g_i}$ and $\mathbf{o_i}$ among sites.

CHAPTER 3

COMBINATORIAL OPTIMIZATION PROBLEMS - AN INTRODUCTION

The fertilizer problem addressed in this work belongs to the general class of problem called Combinatorial Optimization (CO) [5, 55].

Definition 1 *A combinatorial optimization problem* $\mathbf{P} = (\mathbf{X}, \mathbf{D}, f, \mathbf{C})$ *is defined as*

- *A set of discrete variables* $\mathbf{X} = \{x_1, ..., x_n\}$ *representing the problem entity*
- *Variable domain* $\mathbf{D} = \{D_1, ..., D_n\}$ *over* $\mathbf{X}$*. Each* D_i *gives the set of possible value assignments for* x_i
- *The objective function(s)* f *to be optimized, where*

$$f : D_1 \times ... \times D_n \rightarrow \Re^+$$

- *Constraints* $\mathbf{C} = \{C_1, ..., C_m\}$ *over the problem* $\mathbf{P}$*, where*

$$C_i \subseteq D_1 \times ... \times D_n$$

For a given problem $\mathbf{P}$, the solution space $\mathbf{S}$ contains all feasible value assignments for $\mathbf{X}$ that are subject to constraints $\mathbf{C}$

$$\mathbf{S} = \{\ s = \{\ (x_1, v_1), ...,\ (x_n, v_n)\ \}\ |\ v_i \in D_i, s \text{ satisfies constraints } \mathbf{C}\ \}$$

The goal of the optimization problem is to find an element $s^* \in \mathbf{S}$ whose value assignment maximizes[1] the objective function f. *Solution quality* refers to the objective function

---

[1] In the literature, optimization could refer to either maximization or minimization, depending on the given problem definition. For the work in this paper, maximization is used as the default definition.

value ($f(s)$) of a given solution point s, where $s \in \mathbf{S}$. To be more formal, we define the global optimum s^* for a given problem $\mathbf{P} = (\mathbf{X}, \mathbf{D}, f, \mathbf{C})$ as follows:

Definition 2 *A global optimum s^* in solution space $\mathbf{S}$ has the following property:*

$$\forall s \in \mathbf{S},\ f(s^*) \geq f(s)$$

Global optima refers to solution points that are globally best within the whole solution space.

We provide in this chapter a brief discussion of the general area of optimization, then we give an introduction into some well studied problems in the area of combinatorial optimization.

3.1 Optimization in General

Combinatorial optimization belongs to the general problem class of optimization problems. In the literature, optimization usually refers to the general problem-solving process of finding the best solution (global optimum) to achieve a certain goal. Real life optimization problems also contain local optimal solutions. To define local optimum, we first introduce the notion of neighborhood structure:

Definition 3 *A neighborhood structure is a function $N : \mathbf{S} \rightarrow 2^{\mathbf{S}}$ that assigns to every solution point $s \in \mathbf{S}$ a set of neighbors $N(s) \subseteq \mathbf{S}$.*

$N(s)$ is called the neighborhood of solution point s. For a given optimization problem $\mathbf{P}$, we define local optimum as follows:

Definition 4 *Solution point $\hat{s} \in \mathbf{S}$ is a local optimum in $\mathbf{P}$ if*

$$\forall s \in N(\hat{s}),\ f(\hat{s}) \geq f(s)$$

Local optima refers to solution points that are “locally” best among all their direct neighborhood points.

Continuous vs. Discrete Optimization A given optimization problem may assume either continuous or discrete values for its variable domains. Within the context of solving optimization problems by searching, unless special restrictive assumptions are made to the objective function f (e.g., convexity), it is known that continuous global optimization problem is inherently unsolvable in a finite number of steps [44, 6, 11]. For any arbitrary continuous differentiable function f, one cannot verify with certainty that a given solution point s is not the global optimum without evaluating the function in at least one point in every neighborhood N of s. However, for continuous problems where real valued domains are assumed, the choices of neighborhood N are infinite since the distance $\parallel s' - s \parallel$, where $s' \in N(s)$ representing the neighborhood of s, can be arbitrarily small. Thus, it follows that search algorithms designed to solve a global, continuous and general optimization problem would require an unbounded number of steps [11, 44], and we are unable to find the global optimum s^* in finite time. The goal of finding global optimum is often stated as finding a point in

$$N_\epsilon(s^*) = \{s \in \mathbf{S},\ \parallel s - s^* \parallel \leq \epsilon\}$$

By doing this [11], we are transforming the original continuous problem into a discrete global optimization problem by "discretizing" the solution space using an appropriate neighborhood structure N and granularity level[2] ϵ.

However, hard combinatorial optimization problems usually have the feature that a large number of local optima exist. The computational time for solving these problems by searching is often too long for practical use. In practical application, a given solution s might be accepted if it is in S_ρ, where

$$S_\rho = \{s \in \mathbf{S},\ f(s) \geq \rho\}.$$

Here, ρ is called the solution quality threshold.[3] For problems where the global optimum s^* is known, we usually set ρ to a certain percentage of the optimal solution quality (e.g., 98% of $f(s^*)$). For problems where s^* is unknown, ρ is set to a certain percentage of the best known solution quality.

---

[2] Our following discussion also refers to ϵ as the local search landscape granularity.

[3] S_ρ is also referred to as the level set [11].

In the literature, the difference between local versus global optimization concerns problem structure and acceptability of the solution found [60]. In the first case, some problems have the feature that a local optimal solution is at the same time a global optimum (e.g., the convex programming problem); in the latter case, a local optimum might be accepted as long as its objective function value is above a given quality threshold ρ as described previously [11]. From a general point of view, local optimization could be considered a "sub-problem" of the more general case of global optimization problem.

Constrained vs. Unconstrained Optimization One common type of constrained optimization problem is called Constraint Satisfaction Problems (CSP) [31, 50]. In CSP, it is required to find a feasible solution that satisfies all constraints in $\mathbf{C}$, regardless of the objective function f. Thus, the objective function is usually set to empty or defined as a mapping to a constant value. There also exist some techniques that transform a constrained problem into a unconstrained optimization problem [55], then solve it using standard search methods. For unconstrained problems, we have $\mathbf{C} = \emptyset$.

3.2 Combinatorial Optimization Problems

Combinatorial optimization covers a wide range of applications in the fields of Operations Research and Artificial Intelligence. Here we provide a brief introduction into some well studied problems in this area.

3.2.1 Routing Problems

Traveling Salesman Problem (TSP) The TSP [64, 65] is the most well-studied CO routing problem. In TSP, we try to find a shortest path that covers all cities on a map once and only once. TSP provides a standard test bed for various search algorithms. Although TSPs of small sizes were efficiently solved years ago, solving instances of larger sizes still remains a challenge. Various extensions of TSP have been studied, including Asymmetric TSP (ATSP), where distance between two nodes depends on the direction of the path selected; and Euclidean TSP (ETSP), where path length represents ordinary Euclidean distance. Other extensions of traveling salesman problem can be found in [54].

Vehicle Routing Problems (VRP) In VRP, the problem is to determine multiple routes for a number of vehicles [9]. Vehicles are assigned to a set of geographically dispersed cities or customers. The goal is to transport customers with known demands at the lowest routing cost. VRP is a well-known integer programming problem and could be considered as the combination of the Traveling Salesman Problem (TSP) and Bin Packing Problem (BPP). In BPP, the problem is to pack a set of items into a number of bins where the total item weight does not exceed a maximum value [19, 9]. The goal is to put in as many items as possible using the least number of bins.

Other extensions of VRPs have been studied, for example, the Arc Routing Problem (ARP), the Vehicle Routing Problem with Time Windows (VRPTW) and the Multiple Depot Vehicle Routing Problem (MDVRP) [9]. Network Routing Optimization (NRO) also has its important application in the field of telecommunication and packet-switching networks [4].

3.2.2 Scheduling and Resource Allocation Problems

Scheduling usually refers to the problem of resource allocation under constraints. The classical transportation problem is considered to be this type, where we assign goods from a set of warehouses to a number of factories. The goal is to minimize the total transportation cost, while in the meantime satisfying both the warehouse demands and factory capacities.

Scheduling and allocation problems [19] cover a wide range of applications, especially in the field of Operations Research (OR) and Management Science. Examples include activity scheduling, machine scheduling, and project scheduling. One well-studied problem is called the Quadratic Assignment Problem (QAP) [63], where we assign N activities among N different locations. The goal is to minimize the overall transition cost among different activities. The fertilizer problem addressed in this work could also be considered as a resource allocation problem, where a number of market nutrients are delivered to a set of farm sites. Further references regarding scheduling and resource allocation problems can be found in [33].

3.2.3 Multi-objective Combinatorial Optimization Problems (MCOP)

MCOP refers to the problem of simultaneous optimization of several possibly conflicting and incompatible objective functions [43, 8]. For a given MCOP problem $\mathbf{P} = (\mathbf{X}, \mathbf{D}, \mathbf{f}, \mathbf{C})$, the function set $\mathbf{f}$ contains elements that are contradictory to each other. The solution of a MCOP would be a set of non-dominated solution candidates. In Operations Research, managers make decisions from a set of candidate strategies. The choice of one strategy over another represents the trade-off among various business objectives. In MCOP, the optimal solution is referred to as *Pareto Optimality*, or *Pareto Efficiency* [17]. A solution s^* is said to be a Pareto Optimum if there exists no feasible solution s that further improves at least one function value within function set $\mathbf{f}$, without decreasing the function value of another in $\mathbf{f}$. Example MCOPs include Bus Driver Scheduling Problem (BDSP) and Project Portfolio Selection [43].

3.2.4 CO Problems in Software Engineering

Recent work has been extended into the joint area between Software Engineering and Combinatorial Optimization [32, 1, 15]. One of this class of problems considers software testing as an optimization process, and applies various search algorithms to optimize the testing procedure [32]. Another application is called Software Project Scheduling and Management. This class of problems concerns determining “who does what” during the software development life cycle [1]. The major goal is to generate better project schemes and help increase both the work efficiency and product quality. Further references can be found in [14, 21].

CHAPTER 4

METAHEURISTIC ALGORITHMS - AN INTRODUCTION

We provide in this chapter some major concepts in the field of metaheuristics and give an introduction into several state-of-the-art metaheuristic algorithms.

4.1 Neighborhood Structure and Fitness Landscape

Before applying any search algorithm to the target problem, a problem formalization process is usually required, where we transform the target problem into an appropriate mathematical representation. However, there are gaps between problem definition and modeling [53]. There exists a conceptual process that transforms the original problem into an appropriate model in the algorithm. The model definition is algorithm specific: "the solution space, neighborhood structure, and objective function have to be tailored to the chosen metaheuristics" [53]. As shown in Figure 4.1, this poses a potential problem since different algorithms may be associated with different models, making comparisons difficult. Here, we introduce the important notion of *fitness landscape*.

Definition 5 *Fitness landscape is a triple*

$$\pounds = (\mathbf{S}, N, f)$$

$\mathbf{S}$ *represents the solution space;* f *is the fitness function;* N *is the neighborhood function, where*

$$N\text{:}\ \mathbf{S} \rightarrow 2^{\mathbf{S}}$$

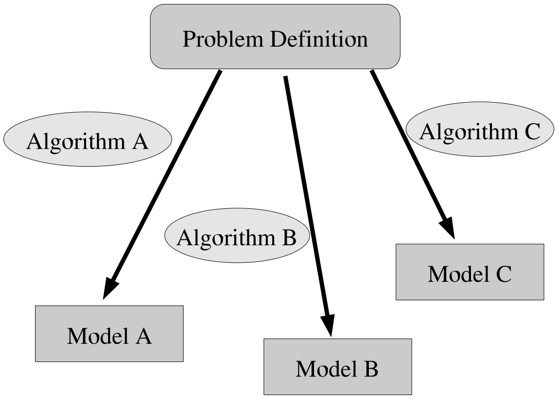

Figure 4.1: The transformation of the problem definition into different model representations.

Conceptually, N defines a neighborhood structure by assigning every solution point $s \in \mathbf{S}$ a set of solutions $N(s)$, where $N(s) \subseteq \mathbf{S}$. $N(s)$ is called the neighborhood of s. Here, we introduce the definition of transition operator φ:

Definition 6 *A transition operator φ maps a given solution point s into a set of solution points $N(s)$ called the neighborhood set of s.*

The notions of the fitness landscape [41, 70, 16] and transition operator enable us to view the search algorithm as the process of exploring a problem-dependent landscape graph. In the graph, *nodes* represent individual solutions; *arcs* represent state transitions where we move from one solution point to another. A *move* is defined as the choice of a solution s' from the neighborhood set $N(s)$ of current point s. One important fact is that, given a problem definition, the choice of the transition operator φ is algorithm-dependent: the *one operator, one landscape* concept [42] states that different algorithms define different transition operators thus searching in different landscapes. Furthermore, this fact implies that algorithm performance is affected by the underlying landscape model it uses. In general, no best choice exists which leads to the best performance over all problem instances. This "empirical result" is also theoretically supported by the *No Free Lunch Theorem* [73]. In no free lunch theorem for search, it is shown that all algorithms, when averaged over all optimization problem instances, perform generally the same; some algorithms may outperform others on one problem but loses its superiority on another problem, and in general, no one algorithm exists that performs best on all problems.

4.2 Metaheuristic Algorithms

Metaheuristic algorithms [5, 37] could be considered as a set of abstract algorithm frameworks. They differ from traditional heuristic methods in that a number of search strategies are integrated into the basic search process. The goal is to explore the solution space in a more effective and efficient manner. The study of metaheuristic algorithms has now become a joint interest in the field of Operations Research, Management Science, and Artificial Intelligence. In recent years, work in this area produced a number of high-performance algorithmic solutions that have successfully solved many hard CO problems.

Here we provide a brief introduction into some major metaheuristics in the literature. We focus on describing a number of local search metaheuristics, but also extend our scope into other advanced techniques including *Ant Colony Optimization*, *Particle Swarm Algorithm*, and *Agent-based Methods*. The introduction is far from exhaustive but aims to provide a sketch of some most active fields and recent advances that are related to our work.

4.2.1 Local Search Metaheuristics

Local Search (LS) is one of the most studied metaheuristics. The search starts with an initial solution, then moves iteratively into a nearby landscape region. Since the next candidate solution is always selected within the current neighborhood set, the search is always based on “local” information.

Local search is also entitled “trajectory methods” [5] since the search process consists of a sequence of continuous trajectories in the solution space. Trajectory here refers to the state transition from the current solution s to its neighbor point s'.

The use of local information inevitably leads to one problem: the algorithm is easily trapped in local optima. An iterative improvement algorithm which accepts only better solution points (strict uphill moves), for example, would stop whenever a local optimum is reached. Although many established local search algorithms have strategies for escaping local optima, finding a good balance between search diversification and intensification still remains a major challenge. Here, *diversification* refers to search behavior that actively explores the solution space without being trapped in some particular regions; *intensification*, on the other hand, usually refers to an intensified search within promising solutions area aiming at reaching high quality solutions.

Iterative Improvement Algorithm (IIA) The Iterative Improvement Algorithm (IIA) moves towards solution points that only improve the best-found solution quality. IIA stops whenever a local optimal point is reached. The algorithm is specified in Algorithm 1.

Algorithm 1 The Simple Iterative Improvement Algorithm

```
Initialize candidate solution $s \leftarrow s_0$
while $s$ is not local optimum do
    Randomly choose a neighbor $s'$ of $s$
    if $f(s') > f(s)$ then
        $s \leftarrow s'$
    end if
end while
return $s$
```

Simulated Annealing (SA) Simulated Annealing is an early metaheuristic algorithm originating from an analogy of how an optimal atom configuration is found in statistical mechanics [46, 35]. It uses temperature as an explicit strategy to guide the search. In Simulated Annealing, the solution space is usually explored by taking random tries. An uphill move is always accepted since it is a better solution. Downhill moves are accepted with a probability that depends on both the current “temperature” level as well as the change in solution quality. Here, change in solution quality is measured by the difference in the objective function value: $f(s') - f(s)$. If we denote the temperature value as T, and the change in solutions quality as ΔE, the probability of accepting a downhill move would be:

$$p = e^{-\frac{\Delta E}{k_B T}}$$

Here, k_B is called the Boltzmann constant, which is used for adjusting the effect of temperature T in computing the downhill probability. A cooling scheme defines how T value drops from high towards low during the search. The basic idea is that we allow more downhill moves at the early stage of the search to keep the algorithm actively exploring the solution space (the higher T, the higher the probability to accept a downhill move); however, as the search continues, downhill moves become less likely so that the search would focus on a limited area that is identified as "good" solution region. Theoretically, given a good cooling scheme that decreases the temperature slowly enough, SA is guaranteed to find the global optimal solution [2, 49].

Algorithm 2 The Simulated Annealing Algorithm

```
Initialize candidate solution $s \leftarrow s_0$, set $s_{best} \leftarrow s$
Initialize cooling scheme, set temperature $T \leftarrow T_0$
while Termination criterion not met do
    Randomly choose a neighbor $s'$ of $s$
    if $f(s') > f(s)$ then
        $s \leftarrow s'$
    else
        $s \leftarrow s'$ with probability $p = e^{-\frac{\Delta E}{k_B T}}$
    end if
    Update temperature $T$ according to the cooling scheme
    if $f(s) > f(s_{best})$ then
        $s_{best} \leftarrow s$
    end if
end while
return $s_{best}$
```

The choice of an appropriate temperature cooling scheme is important and is one of the major meta-parameters[1] to be tuned. Temperature affects the probability of taking a downhill move.[2] The cost of "jumping" out of a local optimal region also depends on how well the cooling scheme fits into the underlying search landscape. Although it is claimed that there exists cooling schemes that guarantee the convergence to global optimum point, the scheme is usually too slow for practical use [35]. Therefore, problem-specific algorithm tuning is required in real applications. The Simulated Annealing algorithm is specified in Algorithm 2. The termination criterion (Algorithm 2, Step 3) is usually defined as reaching an upper bound local search moves.

Tabu Search (TS) Tabu Search [25, 76, 47, 26, 22] differs from Simulated Annealing in the way candidate solutions are selected from the neighborhood set. The previous search history is recorded in the tabu list TL. The algorithm avoids moves toward recently visited areas by using the tabu list to "filter" the current neighborhood set. The filtered neighborhood set forms the *allow set* for the next move:

Definition 7 *An allow set is defined as*

$$N'(s) = N(s) - TL$$

Here, s is the current solution; $N(s)$ is the neighborhood set of s; TL contains recently visited solution points. In the search landscape, the use of tabu list resembles the behavior of laying "land marks" along the search paths to help identify solution regions that are not worth further exploration. Recently proposed algorithms also use long term memory as the strategic guidance for the subsequent search [5]. Information collected during previous search history is used to improve the performance of the algorithm.

Similar to the use of temperature value in Simulated Annealing, the length of the tabu list affects the behavior of the algorithm. A long list represents long-term memory thus forcing the algorithm to explore larger regions of the solution space; a short tabu list, on the contrary, concentrates the search on relatively small solution region. Recently proposed

---

[1] Local search step size is another meta-parameter. Local search step size is discussed in Chapter 5.1.

[2] The Boltzmann constant k_B is another factor affecting the downhill probability.

methods also use a dynamic tabu list scheme, where the tabu list length changes adaptively according to the quality of recently visited solutions [5]. Since the list length balances the effect between intensified regional search and diversified exploration, it is the major meta-parameter to be tuned.

4.2.2 Constructive Local Search

The term *constructive local search* is used here in order to differentiate from the more general class of approaches called *constructive search*. Constructive search refers to the general search technique of generating candidate solutions by iteratively adding solution elements into the partial solution, for example, the classical Dijkstra's algorithm for finding the shortest path in a graph structure. In this section, we are concerned with a more specific approach in which a candidate starting point is constructed prior to local search. The algorithm we study is called Greedy Randomized Adaptive Search Procedure (GRASP) [61, 34, 37]. GRASP consists of two major phases: solution construction and local search. During the first phase, we sample the search space by constructing a set of initial solution candidates. These solutions are then used as the starting points of the subsequent local search. The GRASP algorithm is described in Algorithm 3.

Algorithm 3 Greedy Randomized Adaptive Search Procedure (GRASP)

```
Initialize $s_{best} \leftarrow \emptyset$
while termination criterion not met do
    Construct initial solution $s_0$
    Perform local search on $s_0$ to obtain $s_1$
    if $s_{best} = \emptyset$ then
        $s_{best} \leftarrow s_1$
    else if $f(s_1) > f(s_{best})$ then
        $s_{best} \leftarrow s_1$
    end if
end while
return $s_{best}$
```

The construction phase in GRASP (Step 3, Algorithm 3) uses a greedy randomized construction procedure. In GRASP, each solution s consists of a set of *solution elements* (e.g., individual paths in a route, bits in a string, or entries in a table). The construction begins with an empty partial solution s_0. During each step of the construction, solution elements are ranked and sorted into the *Candidate list (CL)*. The ranked order in *CL* reflects each element's potential contribution to the partial solution quality. A greedy function $g(e)$ is defined to capture this potential contribution to solution quality, and is used to compute the rank value for each element e.

To decide which element to add into the partial solution s_0, we select from *CL* a number of high ranking elements into the *Restricted Candidate List (RCL)*. *RCL* ensures that the final solution built includes some of the most "promising" elements. A selection scheme is used here to specify what elements are selected into *RCL*. We then randomly choose from *RCL* one target element e to add into s_0. The construction repeats until a complete solution s_0 is obtained. This process is described in Algorithm 4.

Algorithm 4 Greedy Randomized Construction

```
Initialize RCL selection scheme and greedy function $g(e)$
Initialize solution $s_0 \leftarrow \emptyset$
while solution $s_0$ not complete do
    Compute the sorted candidate list CL using greedy function $g(e)$
    Compute the restricted candidate list RCL using the RCL selection scheme
    Select element $e$ from RCL randomly. Update $s_0 \leftarrow s_0 \cup \{e\}$
end while
return $s_0$
```

The second phase in GRASP is the local search (Step 4, Algorithm 3). We may choose the basic Iterative Improvement Algorithm (IIA), or more complex schemes such as Simulated Annealing (SA) and Tabu Search (TS). The basic idea in GRASP is to combine heuristic space sampling with local search to conduct an efficient exploration of the solution space. While the construction phase helps identify good solution regions in the search space, the local search enables an intensified exploration of the identified regions. Previous work [61] shows that, given a good combination of construction scheme and local search process, GRASP usually yields high solution quality. To explain the search behavior of GRASP, we first introduce the notions of an *attractor* and a *basin of attraction* [75, 74].

Definition 8 *An attractor* $\hat{a}$ *is a local optimum point in the fitness landscape £. The basin of attraction for* $\hat{a}$ *is the set of all solution points* $s \in \mathbf{S}$ *from which we could reach the attractor* $\hat{a}$ *by applying local search.*

The selection of an appropriate construction scheme is important in GRASP. First, we would like to generate initial points that belong to "promising" solution regions; second, we wish to generate points that are within the basins of attraction of different local optima. The task of search diversification is achieved through a good sampling of the space. Local search takes the role of improving the initial solution by "climbing" the basin until it reaches an attractor. However, for hard optimization problems where a large number of local optima exist, the local search should also be allowed to "move out" of the initial point's basin and perform an intensified search in nearby regions. This improves the chance of finding a higher-quality result. As a consequence, it is crucial to select both a good construction phase and local search that together work well for the given problem instance.

4.2.3 Swarm Intelligence

A new class of metaheuristic algorithms has been recently studied that integrates the idea of swarm intelligence into the algorithm framework design. *Swarm Intelligence* (SI) [45] is a new computational paradigm based on the underlying principle of natural system behaviors. The system consists of many individuals, such as a colony of ants or a flock of birds. Especially in recent years, the growing interest in applying swarm intelligence to CO problem domain has lead to the emergence of two successful metaheuristic frameworks: Ant Colony Optimization (ACO) [13, 12] and Particle Swarm Optimization (PSO) [38, 45, 68, 67]. In these swarm algorithms, each individual uses simple rules to govern their local search activity. The individuals in the swarm interact with each other during the search process to achieve the same optimization goal. Recent work [69, 51, 67, 71] shows that these swarm algorithms are capable of solving many hard CO problems and can outperform some traditional metaheuristics.

Ant Colony Optimization (ACO) The underlying model in the ACO algorithm is called the Ant System (AS) [13, 12], whose overall goal is defined as finding a shortest route between the nest and food resource.[3] The foraging behavior of real ants provides a model to achieve this goal: while walking from nest to food and vice versa, each ant deposits a substance called *pheromone* on the path. When deciding the next direction to search, they choose with higher probability those paths that are marked with stronger pheromone concentration. The amount of pheromone each ant deposits is proportional to its own evaluation of that path's importance (e.g., the path length). As a result, a shorter path will have a greater chance of being visited again by the following ants. This basic behavior is referred to as a *distributed auto catalytic process* [13]. It serves as the foundation for a cooperative interaction within the ant system and usually leads to the finding of the shortest route.

In the ACO algorithm, ants communicate with each other through the amount of pheromone on a path. Each ant incrementally builds its partial solution (a route from nest to food) by exploring the environment and sensing the pheromone along the path. The environment here usually refers to a linked graph. Each node in the graph is associated with an *Ant Routing Table*. The ant at node i chooses the next node j to expand with transition probability p_{ij} as recorded in the routing table. Whenever an ant reaches the food resource, it returns to the nest following its constructed route and updates the pheromone value τ_{ij} along each visited path ($\tau_{ij} \leftarrow \tau_{ij} + \Delta\tau_{ij}$). The amount of pheromone deposited ($\Delta\tau_{ij}$) by the ant is related to the quality of the route found (e.g., the length of the route). The ant routing table on each visited node is also updated according to the changes in pheromone values on all associated paths. The ant may restart the searching process again after returning to the nest and construct another route.

---

[3] Both food resource and nest are represented as nodes in a graph structure in ACO.

Particle Swarm Optimization PSO [45] is a recently proposed population-based metaheuristic framework. Similar to ACO, the algorithm is inspired by the natural behavior of bird flocks searching for food and has been applied for solving various combinatorial optimization problems [67, 71, 38]. The basic ideas in PSO are distributed problem solving and information sharing: search task is dispatched among a swarm of agents called "particles"; each particle acts independently by conducting a local search process in the problem solution space. The best solution found within the whole swarm (global information) is shared among all particles; each particle also maintains the best-solution found by its own search process (local information). In the algorithm, each particle conducts a thorough search within the region between the "global-best" solution point and its own "local-best" solution point. Literature [45, 68] reports that these regions might have a better chance of containing good solutions and the idea of "flying from one good solution location to another" [45] may be a better way of obtaining high-quality result.

Some recent work also views metaheuristic algorithms from the perspective of an agent-based framework [24, 23, 53, 48]. The idea is based on the fact that many state-of-the-art metaheuristic algorithms have encompassed inherent agent-based concepts. The use of agent terminology to describe metaheuristic procedures might help provide a systematic view for understanding the problem solving process of these algorithms. One of these works, named MultiAGent Metaheuristic Architecture (MAGMA) [53], represents a metaheuristic algorithm as a four-level multi-agent architecture. Agent-based cooperative search [10, 62, 71, 36] also draws much attention in the literature for solving combinatorial optimization problems.

4.2.4 Variable Neighborhood Search (VNS)

In VNS [29, 30], we change the search landscape dynamically by switching among different neighborhood structures. The idea is based on the fact that an existing local optimum in one search landscape is not necessarily a local optimum in another. Changing the landscape may help the search process effectively escape from local optima and keep it actively exploring the solution space. One variation of VNS is called *Variable Neighborhood Decomposition Search* (VNDS) [28]. In VNDS, the solution space is decomposed and represented in a set of neighborhood structures. Each neighborhood structure N_i is defined in such a way that several selected dimensions of the solution variables remain constant.[4] The algorithm usually starts by applying a local search (e.g., strict uphill move) within one N_i landscape until a local optimal solution is reached; it then switches to another (e.g., N_j) landscape using a pre-defined switch scheme. VNDS could be considered a space partitioning search technique.

4.3 A Unifying View for Metaheuristic Search

We present in this section an influence diagram that describes the search behavior of the metaheuristic algorithms discussed in this section.

4.3.1 An Influence Diagram

Different algorithms search in different landscapes. For a given problem instance, the underlying fitness landscape is determined by two major factors: *solution space* and *neighborhood structure*. For illustrative purposes, we use the more general term *landscape topology* instead of *fitness landscape*. While *solution space* could be viewed as a set of solution nodes where no connections exist between them, *landscape topology* defines how solution nodes are expanded into their nearby nodes set thus describing the spatial structure of the search landscape. A *search graph*, on the other hand, represents the search history (search path) of one typical run of the algorithm. Different runs of the algorithm may visit a different solution points set, thus having different search graphs.

The *transition operator* φ defines how the search process moves from one solution point to another. Applying φ to the current solution generates a *neighborhood set* containing all candidate solutions for the next move. Different algorithms may define different transition operators, thus having different neighborhood structures. Another factor that differentiates one search algorithm from another is the *solution acceptance scheme*. The

---

[4] This is different from a problem decomposition, since each neighborhood structure (thus its associated landscape) still maintains full features of the original problem, only that some selected dimensions have fixed value during the search.

acceptance scheme affects how the next solution is selected from the neighborhood set. Its non-deterministic feature is the major cause of the variance among different *search graphs*.

To summarize these relations, we present an influence diagram shown in Figure 4.2. In this figure, *solution space* is determined by three factors: the *problem instance*, the *constraints* over the problem, and the *fitness function*. A given candidate algorithm has two major components: *transition operator* and *solution acceptance scheme*. The *landscape topology* defines the spatial structure of the underlying search landscape, and it is determined by both the *solution space* and *neighborhood structure*. Landscape topology is associated with multiple *search graphs*, each representing one typical run of the algorithm. Since the *acceptance scheme* affects how solution points are visited during the search, it is associated with each search graph.

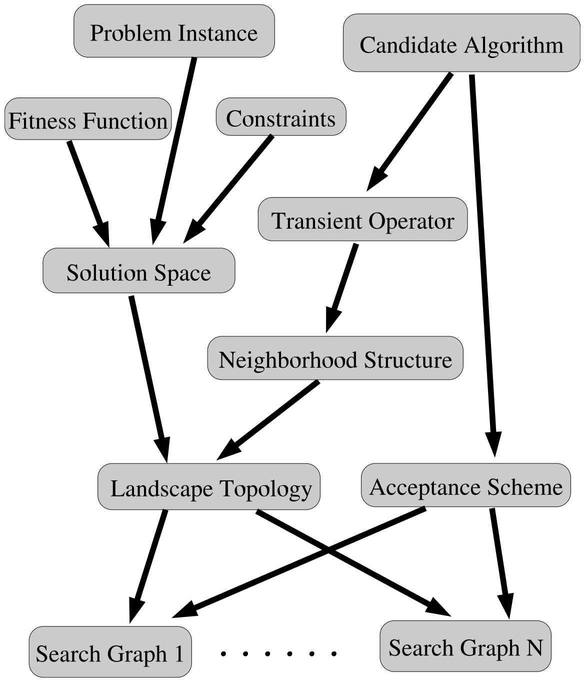

Figure 4.2: A unifying view for metaheuristic search.

4.3.2 Landscape Topology Graph and Algorithm Search Behavior

Figure 4.2 indicates that, for a given problem instance, the search behavior of the metaheuristic algorithm is mainly affected by the underlying landscape topology it uses. Local search, for example, iteratively moves from the current solution point into one of its direct neighbors. Conceptually, the landscape topology in local search could be described using an undirected "Local Search (LS) Graph" (Figure 4.3). In Figure 4.3, nodes represent individual solutions; an edge exists between two nodes if one node could be reached by applying the *transition operator* on the other. Local search graphs have high degree of connectivity, since connections usually occur between two nearby nodes with high probability.[5]

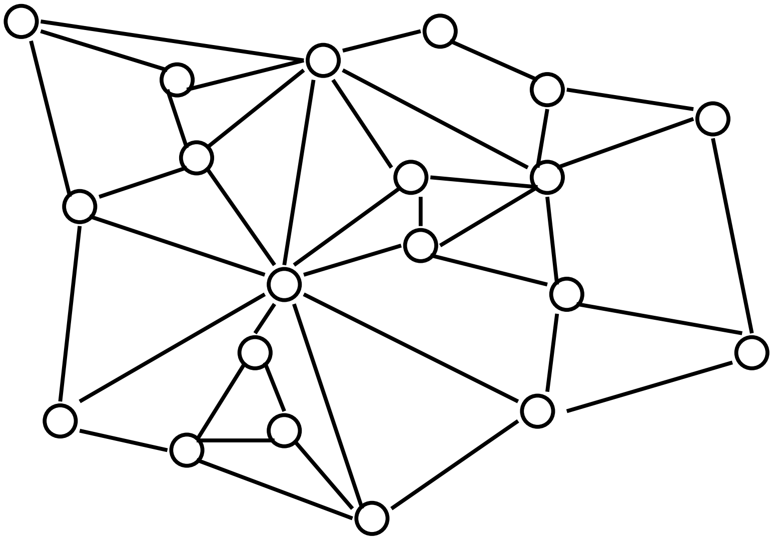

Figure 4.3: A conceptual local search landscape topology graph. Nodes represent solution points. Edges represent possible transitions between solutions (undirected).

Due to the local connectivity feature of the landscape topology, escaping local optima becomes a major issue in local search algorithms. The search process is easily trapped since solution points are locally connected. Jumping out of a local optimal region requires mechanisms such as probabilistic downhill move (e.g., the cooling scheme in SA) or short-

---

[5] Local search topology graph is a static view of the spatial structure of the fitness landscape, and connections between nodes exist throughout the search process of a given algorithm.

term memory (e.g., the tabu list in TS). Local search methods are good at quickly obtaining quality solution points, but require extra runtime cost for actively exploring the whole solution space.

In GRASP, however, the joint effect of space sampling and local search would result in a landscape topology that could be described using a "cluster landscape graph" (Figure 4.4). In Figure 4.4, each cluster has an *attractor*, which is itself a local optimum. The "body" of that cluster is the *basin of attraction* consisting the set of solution points from where a local search could be applied to reach the attractor. The attractor and its basin together define the cluster.

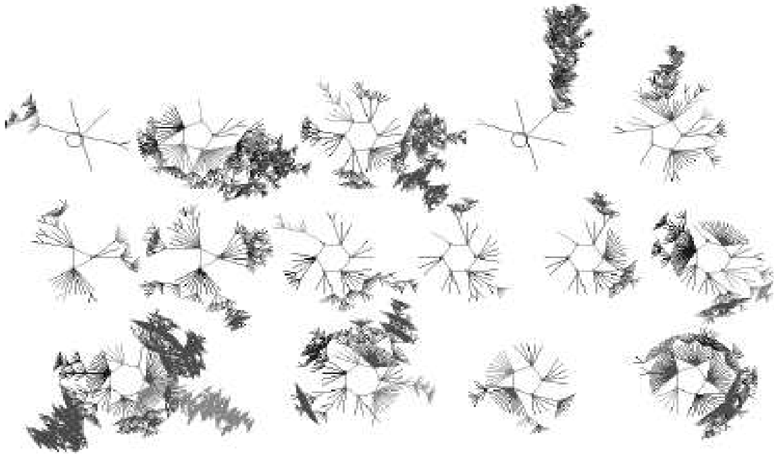

Figure 4.4: A conceptual cluster landscape topology graph. Each cluster represents one local optimal region including an attractor (local optimal point) and the basin (a set of solution points that may lead to the attractor).

The cluster landscape topology indicates that the search behavior in GRASP follows a hierarchical pattern: it first jumps to the attraction basin of one cluster, then climbs up (or down) the basin until it reaches the attractor. The first step requires identifying a "good" cluster in the solution space. In GRASP, the greedy randomized construction process takes this role. GRASP then works on "micro-level" by performing a local search process.

In *Variable Neighborhood Search (VNS)*, however, we switch among different neighborhood structures. The landscape topology in VNS forms a "multi-layer graph". Each layer is a typical “local search graph”, however, different layers now represent different neighborhood structures, thus having different landscapes.

4.4 Algorithm Performance and Performance Measurement

4.4.1 Algorithm Performance

Empirical experiments show that Algorithm Performance ($\mathbf{AP}$) is affected by its Meta-Parameter Setting ($\mathbf{MPS}$) and the target optimization problem $\mathbf{P}_0$. This relation could be described in an influence diagram shown in Figure 4.5.

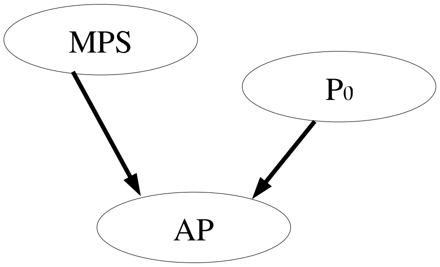

Figure 4.5: Algorithm Performance ($\mathbf{AP}$) is affected by its Meta-Parameter Setting ($\mathbf{MPS}$) and problem instance $\mathbf{P}_0$.

Meta-parameters usually refer to the set of high-level control strategies in various metaheuristic algorithms. Examples would include the cooling schedule in Simulated Annealing and tabu list length in Tabu Search. Algorithm performance generally describes the algorithm’s capability of solving the problem, and it can be measured by a number of performance metrics (e.g., average best-found solution quality, runtime distribution). One major work in our experiment is to tune $\mathbf{MPS}$ in order to obtain an optimal or near-optimal algorithm performance. Optimal algorithm performance achieves either highest best-found solution quality or highest problem solvability. For a given optimal setting

Table 4.1: Algorithm performance table. MPS^*_i specifies the ith element in MPS^*. Each entry in the table is a text description of the best-found parameter setting.

	MPS^*_1	MPS^*_2	...	MPS^*_n
Problem 1	$*$	$*$	...	$*$
Problem 2	$*$	$*$	...	$*$
Problem 3	$*$	$*$	...	$*$

$\mathbf{MPS}^*$ on problem $\mathbf{P}_0$, we are expected to obtain the optimal performance of the algorithm on $\mathbf{P}_0$.

However, since meta-parameters are usually inter-dependent, the tuning work itself might be complex enough to be considered an optimization problem. Systematic mechanisms are usually required to obtain $\mathbf{MPS}^*$. In practice, however, manual tuning obtains only the best-found $\mathbf{MPS}$ that is close to $\mathbf{MPS}^*$.

Furthermore, the optimal meta-parameter setting may vary among different problem instances. To summarize the best-found $\mathbf{MPS}^*$ for each problem, we use the *Performance Table* (Table 4.1).

Table 4.1 could be used by application programmers who implement the algorithm for the specific problem. The variance in $\mathbf{MPS}^*$ over different problem instances also indicates the *robustness* feature of the algorithm; furthermore, the correlation between meta-parameter setting and problem feature could be inferred from this table, for example, how parameter setting changes with respect to problem size.[6] This might help us "predict" potential $\mathbf{MPS}^*$ of the algorithm whenever new problem instances are encountered that have not been studied. New problems might be different in a number of features such as problem size and landscape difficulty. The correlation observed in the tuning work might help us estimate the new optimal meta-parameter settings for the new problem without conducting the whole meta-parameter tuning work again.

One important fact is that in real applications, algorithm performance might also be affected by a number of factors such as program implementation and running environments.

---

[6] To allow this, problem size and various other problem features should be examined as well.

This indicates one potential problem where our best-found $\mathbf{MPS}^*$ in Table 4.1 might not be accurate enough in actual implementation. One solution is to provide graphical user interfaces that allow the manual "micro-tuning" of the suggested meta-parameter values. An alternative way is to provide an automatic tuning mechanism that optimizes algorithm performance for each of the encountered problem instances [40, 39]. However, as mentioned earlier, this brings significant computational cost and is outside the scope of the work in the thesis.

4.4.2 Performance Measurement

The comparative study of metaheuristic algorithms is complicated by the fact that there usually exists a trade-off between solution quality and runtime cost: one algorithm may outperform another given longer runtime bound; on the other hand, algorithms that obtain on-average good solutions may lose their superiority if higher solution quality is required. As a result, the preference of one algorithm over another should be discussed within the context of a given runtime or quality threshold.

A second issue concerning algorithm comparison is that algorithm performance ($\mathbf{AP}$) changes according to different $\mathbf{MPS}$. To compare the performance of two algorithms, we need to first fix each algorithm's $\mathbf{MPS}$ at its best-found $\mathbf{MPS}^*$ level for each encountered problem instance. This gives a "peak" performance comparison of the algorithms. Throughout our empirical study, the following performance measurement methods have been used.

Runtime Quality (RQ) Measurement A RQ graph shows how average best-found solution quality changes over runtime. Solution quality refers to the evaluation function value of the target problem (e.g., total net profit in the fertilizer problem). For each run of the algorithm on a particular problem instance, we record the best-found solution quality at each fixed runtime intervals (e.g., every 1 CPU second, or every 100 local search steps). Average best-found quality at each runtime interval is obtained over multiple runs. Runtime quality measurement shows how the search algorithm converges to good solutions over the entire runtime range. It gives a general picture of the trade-off between solution quality and runtime cost. Comparing RQ graphs for different algorithms shows their different rates of convergence, as well as the best-found quality they are capable of achieving.

Similar to runtime quality, a method called *Quality Runtime (QR) Measurement* is also adopted. In QR, we calculate the average runtime cost r of the algorithm to reach solution quality threshold q_0 (over multiple runs). Runtime is measured in both number of local search moves and CPU seconds (for comparison across algorithms). QR graphs show how the computing cost of the algorithm varies over different solution quality requirements. QR measurement could be used as a supplementary measurement metric for RQ measurement.

Run Time Distribution (RTD) Measurement In the RTD measurement, we measure the probability that the algorithm achieves a given solution quality threshold q_0 over time. The threshold q_0 represents an acceptable solution quality in practical application, and is usually set to a certain percentage of the best-known[7] solution quality (e.g., 98%) for each of the encountered problem instances. Multiple runs of the algorithm are performed, and each run's best-found solution quality is recorded at fixed runtime intervals. The percentage of successful runs whose best-found quality reach q_0 (at each runtime interval) is computed. RTD gives a good measurement of that algorithm's *problem solvability* over runtime. Generally speaking, algorithm whose solvability achieves 1 within an upper runtime bound indicates it is q_0 *complete* for the problem. On the other hand, an algorithm whose solvability fails to reach 1 indicates it is *essentially* q_0 *incomplete* [37].

---

[7] The best-known solution quality is used here since we don't know the global optima for the fertilizer problem.

CHAPTER 5

APPLYING METAHEURISTIC ALGORITHMS TO THE FERTILIZER PROBLEM

Our work applied a number of local search metaheuristic algorithms to the fertilizer problem, including the Iterative Improvement Algorithm (IIA), Simulated Annealing (SA), DNM Local Search and Constructive Local Search (GRASP).

5.1 Iterative Improvement Local Search for the Fertilizer Problem

We applied a simple Iterative Improvement Algorithm (IIA) to the fertilizer problem. As described in Algorithm 1, IIA accepts only uphill moves that improve the best found solution quality. In the fertilizer problem, the solution s consists of both the nutrient mixture table $\mathbf{A}$ and delivery rate table $\mathbf{V}$. The transition operator defines how the search process moves from one solution point to another. In IIA, we make random changes to both table $\mathbf{A}$ and $\mathbf{V}$ together at each local search move. The transition operator in IIA is specified below.

- First, randomly select $\mathbf{a}_l \in \mathbf{A}$, randomly select $a_{il} \in \mathbf{a}_l$, then set $a_{il} \leftarrow a_{il} \pm \Delta a$.[1] Normalize $\mathbf{a}_l$ so that $\sum_{i=1}^{n} a_{il} = 1$. Here, $i = 1, 2, \ldots, n;\ l = 1, 2$.

- Second, randomly select $\mathbf{v}_j \in \mathbf{V}$, randomly select $v_{lj} \in \mathbf{v}_j$, then set $v_{lj} \leftarrow v_{lj} \pm \Delta v$. Here, $l = 1, 2;\ j = 1, 2, \ldots, k$.

---

[1] The selection of $+$ and $-$ is also random.

Tuning local search step size Δa and Δv is one of the major experimental work in our following empirical study. The step size tuning experiment is discussed in Chapter 6.1.1.

5.2 Simulated Annealing Local Search for the Fertilizer Problem

The Simulated Annealing algorithm is described previously in Algorithm 2. In Simulated Annealing, we also make random changes to both table **A** and table **V** together. Thus, the SA algorithm uses the same transition operator as previously described in IIA. The solution acceptance scheme in SA, however, takes a different form: we always accept uphill moves that increase the total net profit (Step 5, Algorithm 2). A downhill move is accepted (Step 8, Algorithm 2) with a probability that depends on both the current "temperature" level T and the change in solution quality. In the fertilizer problem, the change in solution quality is measured by the relative difference in total net profit. We denote P as the original total net profit and $P^{'}$ as the new profit after changes. Assuming $P^{'} < P$, indicating a downhill move, the relative change in solution quality[2] is defined as

$$\Delta E = \frac{P - P^{'}}{P}$$

Thus, the probability of taking a downhill move is

$$p = e^{-\frac{\Delta E}{k_B T}}$$

In Step 10 of Algorithm 2, the *cooling scheme* controls how temperature T changes at each local search move. In the fertilizer problem, two schemes have been examined:

- *Geometric cooling scheme* changes temperature by $T_{k+1} = \alpha \times T_k$, where $\alpha = (\frac{1}{T_0})^{\frac{1}{n}}$, $k = 0, 1, ..., n-1$. Here, T_0 is the initial temperature value; n is the total number of local search moves (the $while$ loop in Algorithm 2); k is the current number of local search moves.

---

[2] The relative change is used here so that ΔE has smaller variance within different local search moves.

- *Arithmetic cooling scheme* changes temperature by $T_{k+1} = T_k - \theta$, where $\theta = \frac{T_0 - 1}{n}$. Here, T_0 is the initial temperature value, n is the total number of local search moves.

In the above cooling schemes, we set the initial temperature $T_0 = 100$ so that the value range of T_k is traversed from T_0 to 1 during the search process. Our preliminary experiments show that Simulated Annealing is a good candidate for solving the fertilizer problem. However, tuning various meta-parameters (e.g., Δa, Δv, k_B and cooling schemes) remains a major challenge. The SA tuning experiment is discussed in Chapter 6.1.

5.3 DNM Local Search for the Fertilizer Problem

An alternative way to solve the fertilizer problem is to consider nutrient delivery as a resource allocation problem. In this work, we proposed a *Delivery Network Model (DNM)* to solve the problem. The basic idea is that, instead of exploring the large space of two tables $\mathbf{A}$ and $\mathbf{V}$ directly, we deliver nutrients incrementally by repeatedly adding "nutrient flows" into the network.

In the network, one set of nodes represents the market nutrients, another set represents the farm sites. A path represents the nutrient flow coming out from a particular nutrient node into a site node. The flow value on path P_{ij} is

$$f_{ij} = a_i v_j,\ a_i \in \mathbf{A},\ v_j \in \mathbf{V}$$

To simplify the description of the model, we assume only one compound blend is used in the fertilizer problem. Thus $\mathbf{A}$ table contains only one column; a_i refers to either a_{i1} or a_{i2} as described in Chapter 2; v_j refers to either v_{1j} or v_{2j} in table $\mathbf{V}$. The model can be easily extended for two compound blends by using two identical delivery networks. Figure 5.1 gives an example of the delivery network with three market nutrients and four field sites. The delivery network model transforms the original problem of finding an optimal delivery scheme into the calculation of a special delivery flow in the network that optimizes the network utility. Utility here refers to the total net profit.

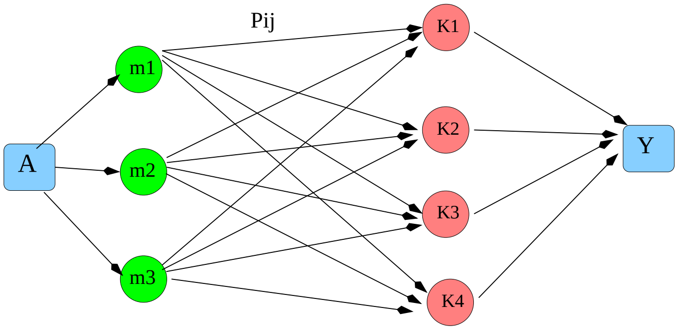

Figure 5.1: A simple delivery network model. m_i represents the ith market nutrient; K_j represents the jth farm sites. Path P_{ij} represents nutrient delivery flow from m_i to K_j. Box A is the decision point where we choose a particular market nutrient m_i to use. Box Y represents the point where we compute the total net profits by summing up the profits of every site K_j.

We implemented a local search algorithm that operates on the network to obtain the optimal flow configuration. The network is first initialized with small flows, then the algorithm iteratively adds nutrients into the network and raises the flow values on some selected paths. We might always accept flow augmentation that increases the network utility. This would result in a strict uphill local search. However, downhill moves may also be accepted as a way of bringing more diversification into the search process.

The termination state is that, after a series of flow augmentations, the algorithm would run into a state where no flow could be added that further increases the network utility.[3] This is the case where the search process is trapped in a local optimum. In this situation, although no larger profit could be obtained by increasing the current flow, it is still possible for us to decrease flows on certain paths and increase the others, resulting in a better solution. To solve the problem, a restart scheme could be applied, where we reset the network flow and restart the augmentation process again. An alternative way is to alter the current flow configuration by allowing "negative flows" into the network so that the local search process is able to jump out of local optima.

---

[3] This is mainly due to the overdose effect in the field, where too much nutrient delivery would cause site output to be reduced.

5.3.1 The DNM Local Search Algorithm

The DNM algorithm is described in Algorithm 5. In Step 4 of Algorithm 5, the flow augmentation process alters the current flow configuration by selectively adding nutrients into the network. This process is described in Algorithm 6.

Algorithm 5 The DNM Local Search Algorithm

```
Initialize network flow $\mathbf{F} \leftarrow \mathbf{F_0}$
Initialize solution $s \leftarrow s_0$, set $s_{best} \leftarrow s$
while Termination criterion not met do
    Perform flow augmentation, update $\mathbf{F}$
    if Overdose effect happens then
        Perform micro tuning, update $\mathbf{F}$
    end if
    Compute candidate solution $s$ from $\mathbf{F}$
    if $f(s) > f(s_{best})$ then
        $s_{best} \leftarrow s$
    end if
end while
return $s_{best}$
```

For the fertilizer problem, we implemented two types of nutrient selection schemes (Step 1, Algorithm 6): random nutrient selection and adaptive nutrient selection. In the random scheme, we select one nutrient m_o randomly and perform the flow augmentation. In the adaptive scheme, each market nutrient is associated with a heuristic value η_o. Similar to the ant colony algorithms (Chapter 4.2.3) where good choices are marked with high pheromone value, the adaptive scheme enables market nutrients that are previously shown "good" to be more likely selected in the future. The probability of selecting nutrient m_o is

$$p_o = \frac{\eta_o}{\sum_{i=1}^{n} \eta_i}$$

Here η_i is the heuristic value associated with m_i; the heuristic value η_o is updated in the following two ways:

Algorithm 6 The Flow Augmentation Process

```
Require: Current network flow $\mathbf{F}$
Require: Augmentation step size $\delta$
1: Probabilistically select nutrient $m_o$ from $\{m_i\}$
2: Increase all nutrient flows going out of $m_o$
   $\{ f_{oj} \leftarrow f_{oj} + \delta v_j \}, v_j \in \mathbf{V}$
3: if Acceptance condition satisfied then
4:     Update $\mathbf{F}$
5: end if
6: return F
```

- *Linear Heuristic Scheme:* if total profit increases by using m_o, then $\eta_o \leftarrow \eta_o + \Delta$, otherwise $\eta_o \leftarrow \eta_o - \Delta$
- *Proportional Heuristic Scheme:* if total profit increases by using m_o, $\eta_o \leftarrow \eta_o \times \theta$, otherwise $\eta_o \leftarrow \eta_o/\theta$

In step 2 of Algorithm 6, f_{oj} is the flow value on path P_{oj} representing the amount of nutrient flow that comes out of nutrient m_o into farm site K_j. Value v_j is the nutrient delivery rate at site K_j. The operation in step 2 ensures that the flow value consistency is maintained among all nutrient flows that come out of the same nutrient.[4]

The solution acceptance scheme (Step 3, Algorithm 6) is described in Table 5.1. Here, a flow augmentation that increases the network profit (uphill move) is always accepted; however, a downhill move indicates the situation where an overdose happens at some farm sites. To alleviate the overdose effect and avoid the algorithm being trapped in a local optimum, we perform the *micro tuning* operation whenever this happens (Step 6, Algorithm 5). The micro tuning process is specified in Algorithm 7.

Here in Algorithm 7, f_{ij} is the flow value on path P_{ij} representing the amount of flow going from nutrient m_i into site K_j. Value a_i is the percentage value of using market nutrient m_i in mixture table $\mathbf{A}$. The micro tuning process decreases the nutrient delivery

---

[4] In DNM, all flows coming out of nutrient m_o use the same a_o to calculate the flow value. Augmenting flow on one flow path without changing the others will cause an invalid a_o to be computed.

Table 5.1: A decision table for the acceptance condition in Algorithm 6, Step 3.

	Decision
Profit Increased	Accept
Profit Decreased	Reject (Overdose)

Algorithm 7 The Micro Tuning Process

```
Require: Current network flow $\mathbf{F}$
Require: Tuning step size $\alpha$
for Each farm site $K_j$ do
    if Overdose effect observed at $K_j$ then
        Reduce all nutrient flows going into site $K_j$ using
        $\{f_{ij} \leftarrow f_{ij} - \alpha a_i\}$, $a_i \in \mathbf{A}$
    end if
end for
return F
```

rates at some farm sites and enables the search to move out of local optimal areas thus being able to explore other regions of the solution space. Our empirical results show that using micro-tuning process effectively improves the best-found solution quality.

5.4 Constructive Local Search for the Fertilizer Problem

We also applied Greedy Randomized Adaptive Search Procedure (GRASP) to the fertilizer problem. The GRASP algorithm is previously described in Chapter 4.2.2. There are two major phases in GRASP: solution construction and local search. The construction phase begins by first initializing an empty partial solution $s_0 = \emptyset$, then iteratively adding solution elements into s_0 until a complete solution is obtained. Multiple runs of the construction would generate a set of solution points. These solutions are then used as the initial search points of the subsequent local search algorithm.

5.4.1 Solution Element and Greedy Function g(e)

In the fertilizer problem, solution s consists of two tables: mixture table $\mathbf{A}$ and delivery table $\mathbf{V}$. The solution elements in the fertilizer problem refer to different value assignments of entries of table $\mathbf{A}$ or $\mathbf{V}$. Let $\mathbf{C}$ be the value domain over variable a_{il}, where $a_{il} \in \mathbf{A}$; let $\mathbf{D}$ be the value domain over variable v_{lj}, where $v_{lj} \in \mathbf{V}$. The solution element is defined as:

$$e \in \{a_{il} = c | c \in \mathbf{C}\} \bigcup \{v_{lj} = d | d \in \mathbf{D}\}$$

In the fertilizer problem, since each solution variable (a_{il} or v_{lj}) may take a number of discrete value assignments, the total number of solution elements is

$$|\mathbf{C}| * |\mathbf{A}| + |\mathbf{D}| * |\mathbf{V}|$$

Since the continuous spaces of $\mathbf{C}$ and $\mathbf{D}$ are discretized at a certain granularity level, the degree of granularity directly affects the computing cost of the GRASP construction process. The construction phase might be computationally costly if too small granularity is used (thus more value choices for each variable). We found that for $\mathbf{C}$ with value range

$[0, 1]$, a granularity of 0.01 is usually appropriate (thus leading to approximately 100 value choices for each $a_{il} \in \mathbf{A}$). For $\mathbf{D}$ with value range $[40, 200]$ as constraint,[5] a granularity of 10 is appropriate.

The greedy function $g(e)$ in GRASP is used for ranking a given solution element e. The function provides a heuristic estimation of using e to update the current partial solution s_0. Thus, the greedy function considers both the value of e and s_0 together. In the fertilizer problem, our greedy function takes the following form:

$$g(e) = f(s_0 \cup \{e\}) - f(s_0)$$

Here s_0 represents the partial solution under construction; e is the solution element being evaluated and $f(s_0)$ refers to the total net profit P of s_0 in the fertilizer problem. However, since s_0 is a partial solution, there are certain variables in s_0 (e.g., $a_{il} \in \mathbf{A}$ or $v_{lj} \in \mathbf{V}$) whose values have not been determined during the construction. To represent an incomplete solution s_0 and allow the evaluation of its function value, we initialize all variables in the table $\mathbf{A}$ and $\mathbf{V}$ with minimum allowable values. Adding an element e into s_0 is then equivalent to the operation of setting the corresponding variable to that element's value.

5.4.2 Restricted Candidate List (RCL) and RCL Selection Scheme

The restricted candidate list contains a set of promising solution elements that could be used for solution construction. The selection scheme defines how elements are selected into *RCL* from candidate list *CL*. In a value-based scheme, we select elements from *CL* that satisfy the following property:

$$\forall e \in \mathit{CL},\ g(e) \geq g_{max} - \alpha(g_{max} - g_{min})$$

where

$$g_{max} = \max\{g(e) | e \in \mathit{CL}\},$$

$$g_{min} = \min\{g(e) | e \in \mathit{CL}\}$$

---

[5] The farm machinery has constraint that the delivery rate can only be set to within a certain quantity range.

Parameter α controls the number of elements that are selected into *RCL* and affects the quality of the constructed solution. The construction phase with $\alpha = 0$ is equivalent to a greedy process where only the best elements are used; larger α leads to a more random element selection process.

The selection of local search procedures also has an important effect on the overall performance of GRASP. Three different local search algorithms are applied including *Simulated Annealing*, *DNM Local Search*, and *Iterative Improvement Algorithm*.

5.5 Other Metaheuristic Algorithms

We also attempted to apply Ant Colony Optimization (ACO) algorithm to the fertilizer problem. In ACO, the optimization goal is defined as finding the shortest route (global optimum) between the nest and food resource. However, the challenge is to transform the fertilizer problem into an appropriate graph structure representation. ACO is usually considered a construction method, since the search begins with an initial empty solution (an empty route, starting from the nest) and terminates whenever a complete solution is obtained (a complete route, ended at food resource). The high-dimensional solution space in the fertilizer problem (e.g., $\mathbf{A}$ table and $\mathbf{V}$ table) complicates the issue since the number of potential nodes and paths in the graph structure might be very large, making the representation of the fertilizer problem inefficient.

Some other thoughts include applying Evolutionary Algorithms, for example, Genetic Algorithm (GA) [72] to the problem. In GA, solutions are first encoded in strings; we then iteratively apply to the strings a series of operators including crossover, mutation and selection. In the crossover operation, two selected parent strings are used to produce the child string; the mutation operation is similar to a local search process, where alterations are made to the child string after crossover; in the selection operation, we strategically pick a number of strings having high evaluation function value (e.g., the total net profit in the fertilizer problem) and promote them into the next strings generation. The whole process repeats until a given criterion is met, for example, the average fitness value of the string population is above a given threshold. Although string encoding is rather straightforward for the fertilizer problem (e.g., directly view table $\mathbf{A}$ and table $\mathbf{V}$ as strings), the large uncertainty involved in defining and specifying various operators is a major challenge. From the perspective of metaheuristic algorithm, GA can be considered a “hybrid” algorithm combining local search (mutation), solution set filtering (selection), and construction method (crossover). Finding an appropriate combination of different problem-solving strategies that together work well and lead to the convergence of good solutions requires significant experimental effort.

CHAPTER 6

EMPIRICAL RESULTS

The algorithms we studied include: Simulated Annealing (SA), DNM Local Search, Iterative Improvement Algorithm, and GRASP. The empirical experiment contained two parts: we first tuned each of the studied algorithms to obtain their near-optimal meta-parameter settings. The performance of the algorithm under different meta-parameter settings were evaluated and compared to each other. We then conducted a comparative study of these algorithms. Each experiment focused on analyzing certain aspects of the algorithm and all experiments were run against three instances of the fertilizer problems:

- 40-site Canola instance
- 40-site Wheat instance
- 100-site Wheat instance

The 40-site Canola and 100-site Wheat are considered two different problems since the data are obtained from two different farm fields. The 40-site Wheat is a subset of the 100-site Wheat data. In order to compare algorithm performance on problem instances of the same size, we randomly picked 40 sites from the larger 100-site wheat and obtained the 40-site Wheat instance. Our experiments were run on a PC with double 1.80 GHz Pentium IV CPU, 1024KB cache, 1024 MB RAM and Mandrake Linux 10.2. One typical run of the algorithm generally takes 30 CPU seconds on 40-site problems with fixed local search moves n at $160,000$. On the 100-site problem, this time is approximately 180 CPU seconds. Each basic measurement (e.g., quality runtime measurement, runtime distribution measurement) was based on 100 independent runs of the algorithm and all measurements were tested against the three studied problem instances.

6.1 Simulated Annealing

6.1.1 Landscape Granularity

Our first experiment adjusted the local search move size (Chapter 5.1) to obtain a good neighborhood structure. Tuning step sizes Δa and Δv requires comparing algorithm performance under different value settings. Since these two parameters control the granularity of the local search landscape, they affect the performance of the algorithm significantly. To estimate an appropriate value range for Δa and Δv, we first note the constraints $a \in [0, 1]$ and $v \in [40, 200]$. Our preliminary experiments found a reasonable value range of $[0.001, 0.01]$ for Δa and $[1, 10]$ for Δv. We tested a number of (Δa, Δv) pairs and computed for each pair the average best-found solution quality of the algorithm over 100 runs. To eliminate the effect of the temperature cooling schedule and the Boltzmann constant setting, no downhill moves were used in this tuning.[1] The empirical results are summarized in Table 6.1

Table 6.1 shows that the optimal granularity are found within range $[0.001, 0.005]$ for Δa and $[5, 10]$ for Δv over all problem instances. It is also observed that changes in Δa have a larger effect over solution quality and generally speaking, Δa requires smaller granularity. For example, a series of good settings are found along the column at $\Delta a = 0.001$. The explanation is as follows: since the mixture table is used for producing the two compound blends for all farm sites, small changes in table $\mathbf{A}$ would result in an accumulated large changes in nutrient delivery at all farm sites. The empirical result in Table 6.1 suggests that the mixture table $\mathbf{A}$ may require a more detailed search, using finer granularity, than the delivery rate table $\mathbf{V}$.

To see how the granularity setting affects the runtime performance of the local search, we selected 9 (Δa, Δv) pairs and conducted the *RunTime Distribution (RTD)* measurements showing how the problem solvability of the algorithm increases over runtime. Problem solvability here refers to the probability of the algorithm in reaching a given solution

---

[1] Under this setting, the algorithm is identical to the Iterative Improvement Algorithm (IIA) where only strict uphill move is allowed.

Table 6.1: Local search step size tuning matrix. Each entry gives the average found solution quality (total net profit) of SA under different granularity settings, as well as the standard deviation and problem solvability (with respect to 98% of the best-known solution quality). Bolded text indicates the best and second-best solution quality.

(a) 40-site Canola problem, best-known quality q = 7280

Δv / Δa	0.001	0.005	0.01
1	$7166_{\pm 32}$(82%)	$7130_{\pm 24}$(44%)	$7130_{\pm 28}$(42%)
5	$\mathbf{7212}_{\pm \mathbf{37}}$(99%)	$7172_{\pm 35}$(92%)	$7159_{\pm 31}$(84%)
10	$\mathbf{7235}_{\pm \mathbf{40}}$(99%)	$7198_{\pm 37}$(99%)	$7178_{\pm 31}$(96%)

(b) 40-site Wheat problem, best-known quality q = 2780

Δv / Δa	0.001	0.005	0.01
1	$2726_{\pm 24}$(65%)	$2711_{\pm 27}$(36%)	$2701_{\pm 30}$(24%)
5	$\mathbf{2734}_{\pm \mathbf{22}}$(72%)	$2723_{\pm 21}$(62%)	$2716_{\pm 23}$(44%)
10	$\mathbf{2732}_{\pm \mathbf{21}}$(68%)	$2728_{\pm 22}$(73%)	$2718_{\pm 24}$(50%)

(c) 100-site Wheat problem, best-known quality q = 7100

Δv / Δa	0.001	0.005	0.01
1	$6940_{\pm 47}$(36%)	$6894_{\pm 63}$(4%)	$6861_{\pm 87}$(1%)
5	$\mathbf{6974}_{\pm \mathbf{29}}$(77%)	$6939_{\pm 59}$(44%)	$6898_{\pm 72}$(10%)
10	$\mathbf{6974}_{\pm \mathbf{34}}$(78%)	$6954_{\pm 49}$(64%)	$6927_{\pm 59}$(29%)

solution threshold. The results are shown in Figure 6.1, Figure 6.2 and Figure 6.3.

Our observation is that large granularity generally speed up the algorithm's converging process to good solutions, raising the problem solvability during the early time of the search; however, it also suffers from an earlier performance stagnation (e.g., curves in Figure 6.2 and Figure 6.3 at Δa = 0.01 and Δv = 10), where the problem solvability ceases to increase even though longer runtime is given. Our explanation for this phenomenon is as follows: since a small granularity setting (small search step) discretizes the continuous solution space of the fertilizer problem into a large number of discrete points, it considerably increases the runtime cost of the local search for traversing the search space thus impeding the finding of good solutions. A large granularity setting, on the other extreme, enables the algorithm to quickly reach the good solution region by taking large steps;

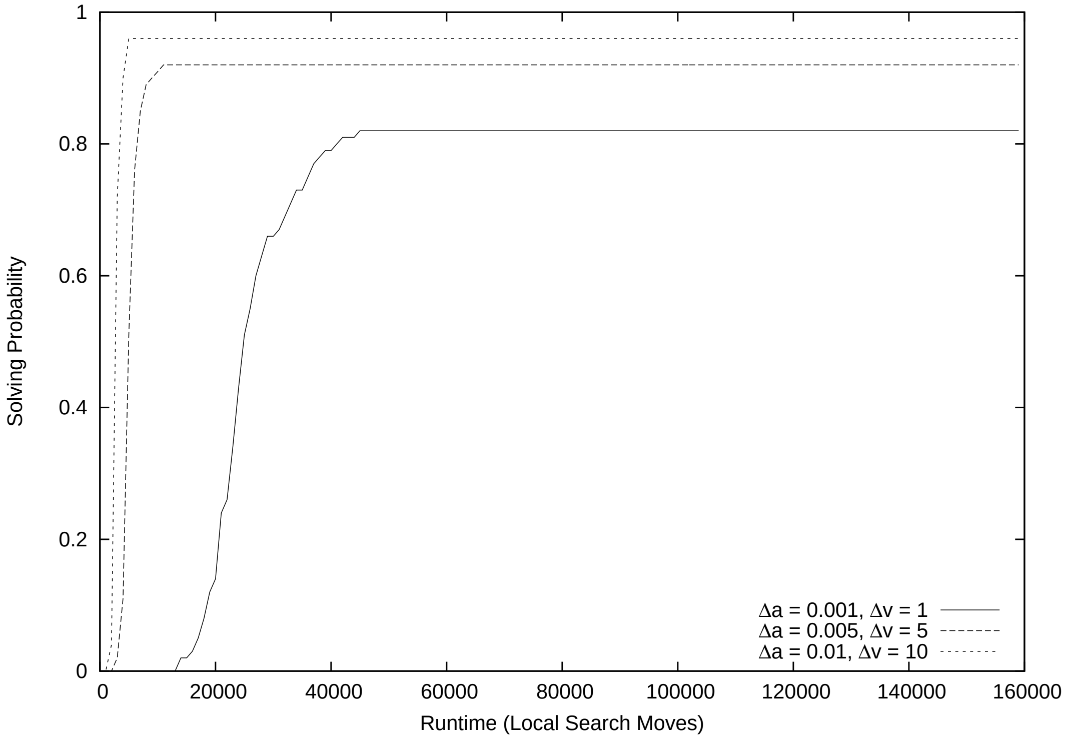

Figure 6.1: RunTime Distribution (RTD) comparison of different SA granularity settings on the 40-site canola problem. Δa is the local search step size for changing $\mathbf{A}$ table; Δv is the local search step size for changing $\mathbf{V}$ table. The quality threshold is set to 98% of the best known solution quality (q = 7280) for the problem.

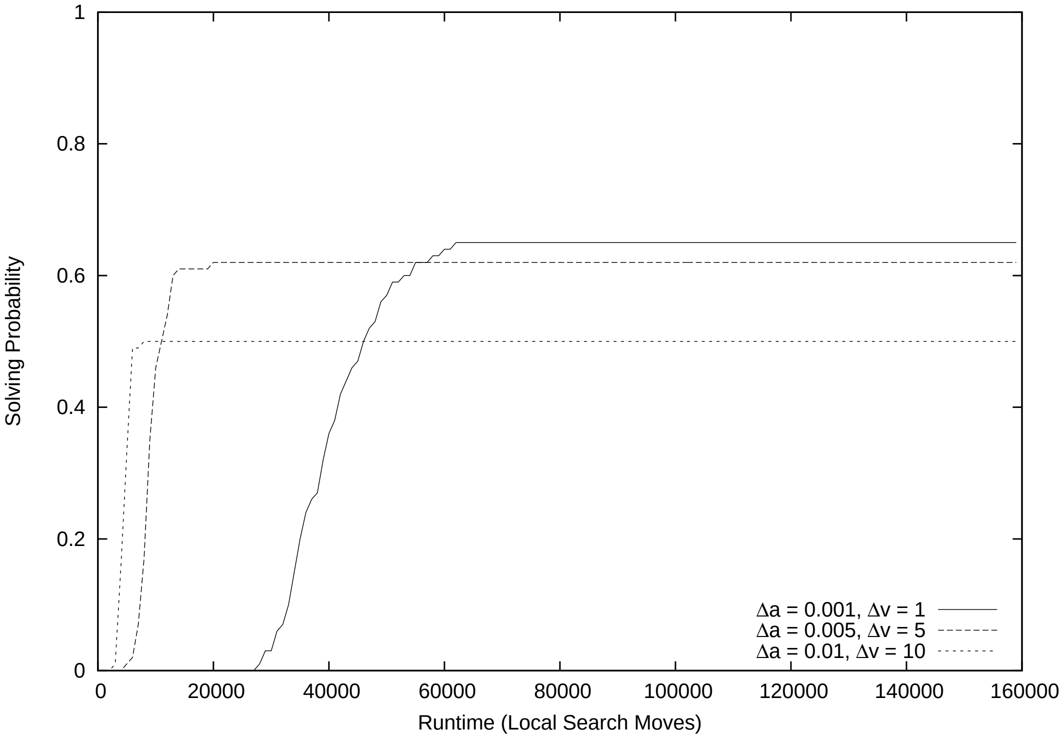

Figure 6.2: RunTime Distribution (RTD) comparison of different SA granularity settings on the 40-site wheat problem. Δa is the local search step size for changing $\mathbf{A}$ table; Δv is the local search step size for changing $\mathbf{V}$ table. The quality threshold is set to 98% of the best known solution quality (q = 2780) for the problem.

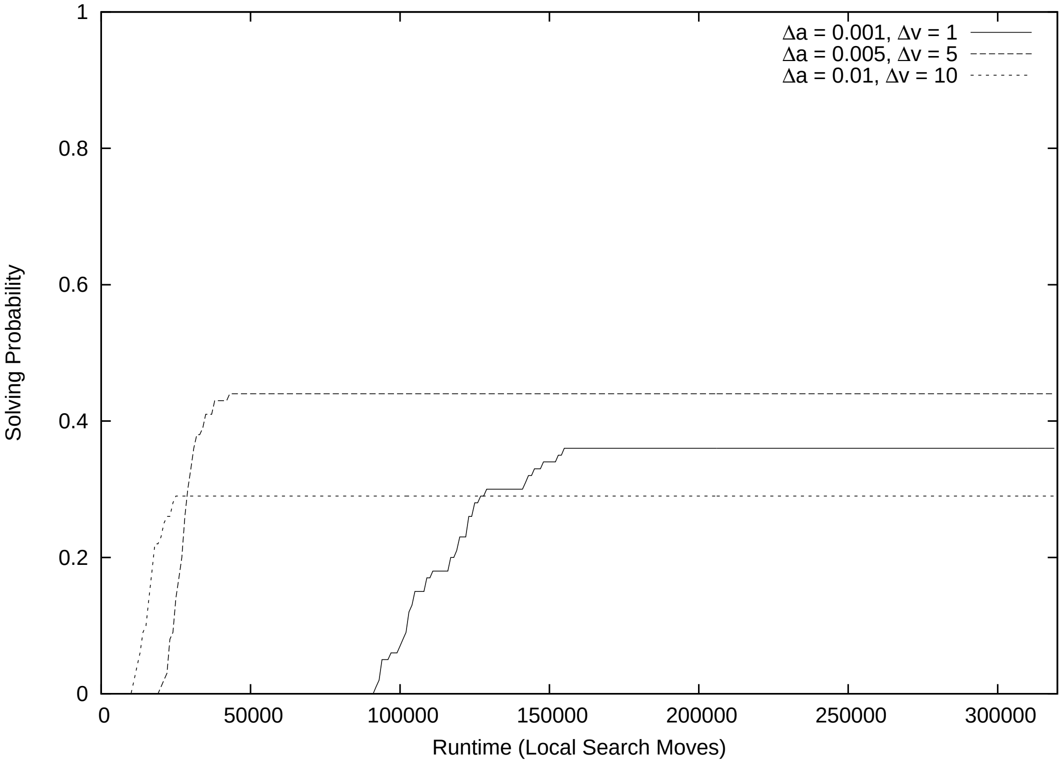

Figure 6.3: RunTime Distribution (RTD) comparison of different SA granularity settings on the 100-site wheat problem. Δa is the local search step size for changing $\mathbf{A}$ table; Δv is the local search step size for changing $\mathbf{V}$ table. The quality threshold is set to 98% of the best known solution quality (q = 7100) for the problem.

however, it has the disadvantage of bypassing a number of potentially good local optimal solutions, resulting in an earlier performance stagnation and low problem solvability to be observed by the end of the search. An optimal step size achieves the balance between search efficiency and result quality by using an appropriate landscape granularity level for the underlying problem instance.

6.1.2 Downhill Move Scheme

In this section, we compared algorithm performance under different downhill move schemes. In SA, since both the temperature cooling schedule and the Boltzmann constant k_B affect the probability of taking downhill moves, we performed a combinatorial examination of both these two parameters together. For each of the two cooling schedules (*Geometric scheme* and *Arithmetic scheme*), we measured algorithm performance under a series of k_B values. The experimental results are shown in Table 6.2, Figure 6.4, Figure 6.5 and Figure 6.6.

Our first observation from Table 6.2 is that there exists an optimal k_B value for each cooling scheme that achieves highest solution quality, and this value does not vary significantly across fertilizer problem instances. For the arithmetic scheme, we found the optimal value at $k_B = 0.005$; for the geometric scheme, optimal value is found near $k_B = 0.05$. Similar optimal value ranges are also shown in Figure 6.4, Figure 6.5 and Figure 6.6. Our second observation is that the arithmetic scheme outperforms the geometric one for small k_B values (e.g., $k_B = 5 \times 10^{-5}$), but loses its superiority for large k_B (e.g., $k_B = 0.5$). The optimal k_B shifts to large value under geometric scheme and equivalent optimal performance is observed using either arithmetic cooling schedule with $k_B = 0.005$, or geometric cooling schedule with $k_B = 0.05$.

Our explanation for the first observation is as follows: the Boltzmann constant k_B controls the value scale for the temperature thus affecting the probability of taking a downhill move; a small k_B value indicates less chance of taking a downhill move thus representing a strong intensification of the search strategy. The local search might be easily trapped in local optima and fail to reach the given quality threshold. On the other hand, a large k_B value allows more downhill moves to be taken thus increasing the probability of finding

Table 6.2: Solution quality comparison. Each entry gives the average best-found solution quality (total net profit) of the algorithm under different downhill move schemes. Both quality standard deviation and problem solvability (with respect to 98% of best known solution quality) are given. Bolded text indicates the optimal quality under each cooling schedule. The best-known solution quality was obtained from preliminary experiments on each studied problem instance, and was exceeded during the tuning experiment (e.g., the 40-site canola problem).

(a) 40-site Canola problem, best-known quality $q = 7280$

Cooling/ $\mathbf{k_B}$	5×10^{-5}	5×10^{-4}	5×10^{-3}	5×10^{-2}	5×10^{-1}
Arithmetic	$7226_{\pm 33}(100\%)$	$7253_{\pm 29}(100\%)$	$\mathbf{7312}_{\pm \mathbf{24}}(100\%)$	$7306_{\pm 24}(100\%)$	$7152_{\pm 112}(92\%)$
Geometric	$7217_{\pm 34}(100\%)$	$7239_{\pm 29}(100\%)$	$7298_{\pm 26}(100\%)$	$\mathbf{7315}_{\pm \mathbf{20}}(100\%)$	$7281_{\pm 28}(100\%)$

(b) 40-site Wheat problem, best-known quality $q = 2780$

Cooling/ $\mathbf{k_B}$	5×10^{-5}	5×10^{-4}	5×10^{-3}	5×10^{-2}	5×10^{-1}
Arithmetic	$2737_{\pm 18}(78\%)$	$2756_{\pm 25}(93\%)$	$\mathbf{2777}_{\pm \mathbf{9}}(99\%)$	$2770_{\pm 9}(100\%)$	$2654_{28}(0\%)$
Geometric	$2737_{\pm 18}(77\%)$	$2751_{\pm 14}(94\%)$	$\mathbf{2776}_{\pm \mathbf{9}}(99\%)$	$2775_{\pm 15}(98\%)$	$2751_{\pm 14}(94\%)$

(c) 100-site Wheat problem, best-known quality $q = 7100$

Cooling/ $\mathbf{k_B}$	5×10^{-5}	5×10^{-4}	5×10^{-3}	5×10^{-2}	5×10^{-1}
Arithmetic	$6983_{\pm 24}(86\%)$	$7028_{\pm 23}(98\%)$	$\mathbf{7057}_{\pm \mathbf{49}}(98\%)$	$7041_{\pm 42}(97\%)$	$6749_{\pm 85}(0\%)$
Geometric	$6973_{\pm 45}(79\%)$	$7006_{\pm 25}(96\%)$	$7059_{\pm 24}(99\%)$	$\mathbf{7064}_{\pm \mathbf{15}}(99\%)$	$6982_{\pm 47}(84\%)$

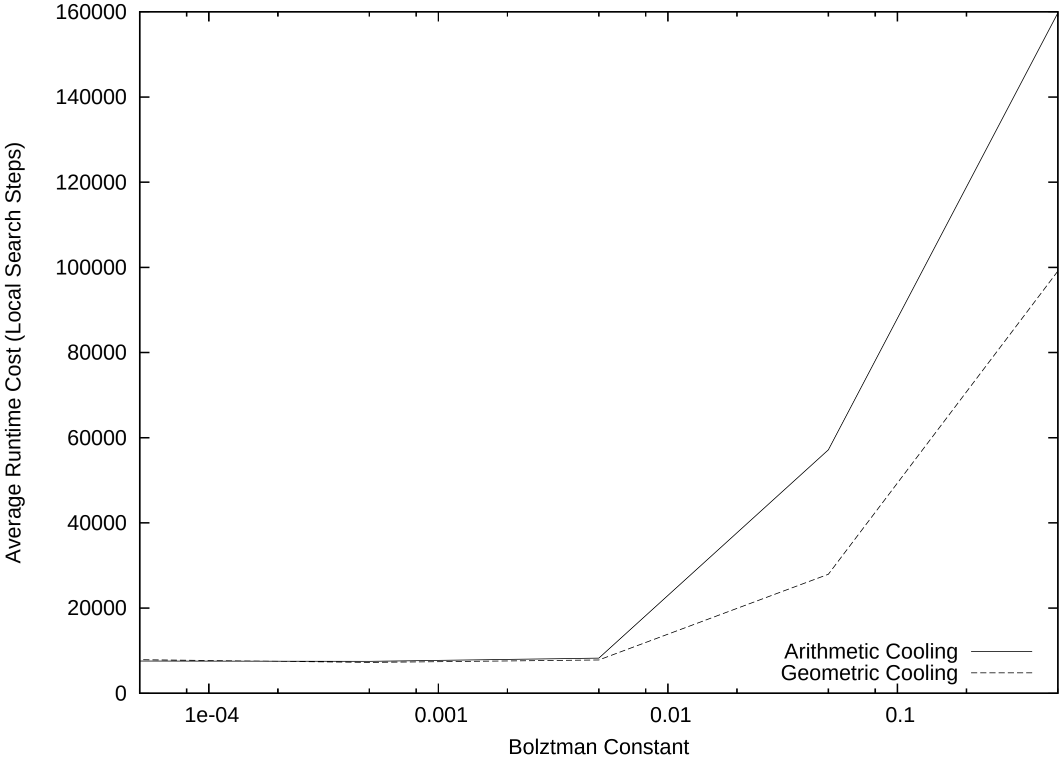

Figure 6.4: Downhill move experiment on the 40-site Canola problem, showing the average runtime cost of the SA algorithm to reach a given solution quality threshold under different Boltzmann constant settings and cooling schedules. The threshold we use is 98% of the best known solution quality ($q = 7280$). The granularity setting is fixed at $\Delta a = 0.001,\ \Delta v = 5$. Runtime is measured in terms of number of local search moves.

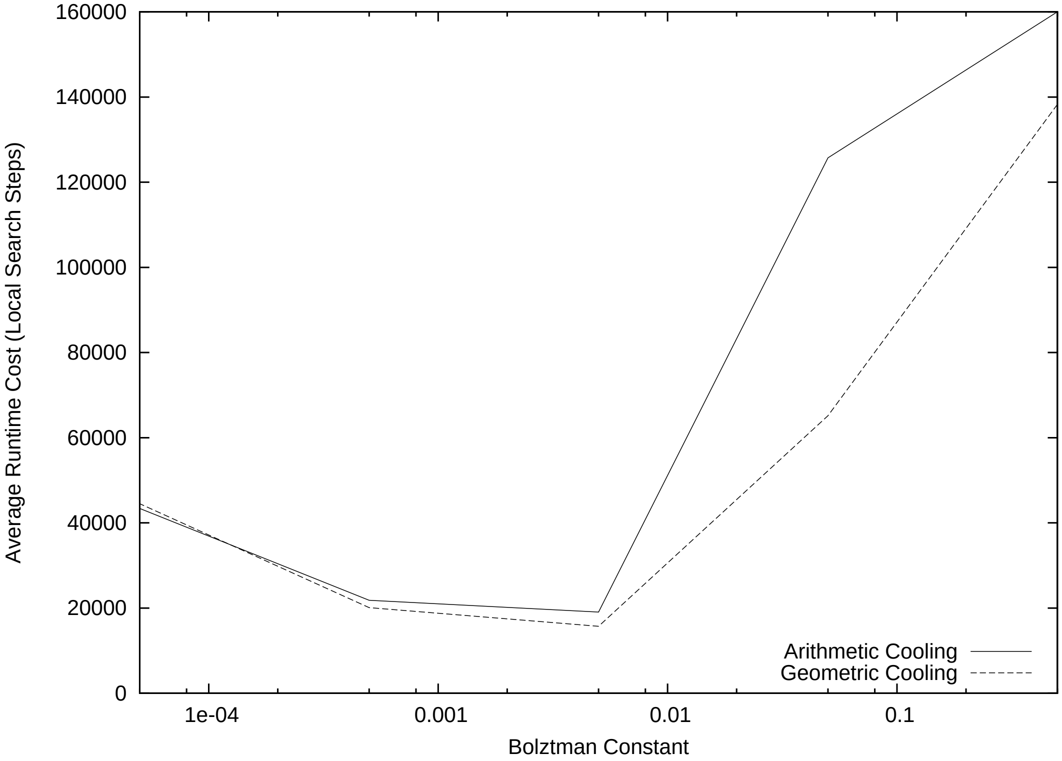

Figure 6.5: Downhill move experiment on the 40-site Wheat problem, showing the average runtime cost of the SA algorithm to reach a given solution quality threshold under different Boltzmann constant settings and cooling schedules. The threshold we use is 98% of the best known solution quality ($q = 2780$). The granularity setting is fixed at $\Delta a = 0.001,\ \Delta v = 5$. Runtime is measured in terms of number of local search moves.

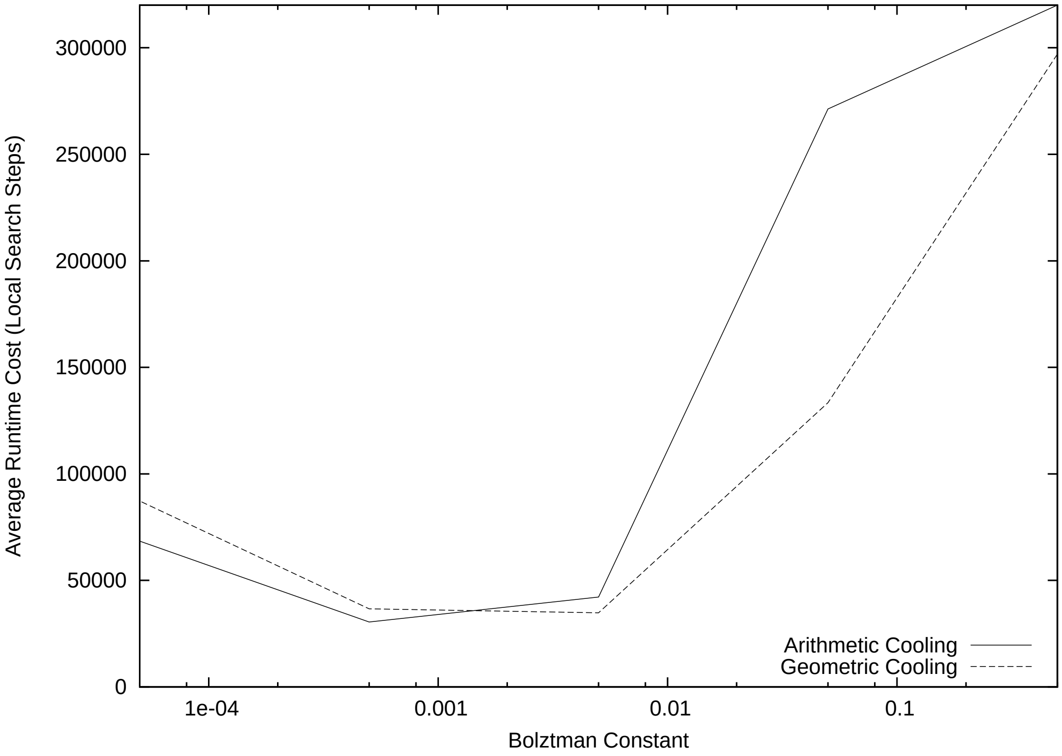

Figure 6.6: Downhill move experiment on the 100-site Wheat problem, showing the average runtime cost of the SA algorithm to reach a given solution quality threshold under different Boltzmann constant settings and cooling schedules. The threshold we use is 98% of the best known solution quality ($q = 7100$). The granularity setting is fixed at $\Delta a = 0.001,\ \Delta v = 5$. Runtime is measured in terms of number of local search moves.

better solutions. However, since the benefit of taking downhill moves requires longer runtime to manifest, the average runtime cost to reach the given quality threshold increases significantly (e.g., shown in Figure 6.4 at $k_B = 0.5$). In such situations, the algorithm either solves the problem using longer runtime, or fails to solve it within the experimental runtime bound. Thus, an optimal k_B value achieves the balance between computational cost and solution quality.

Our intuitive explanation for the second observation is as follows: there seems to exist a constant amount of downhill moves required in the problem that allows the algorithm to achieve an optimal performance using different downhill scheme combinations.[2] Since the temperature level drops more slowly in the arithmetic cooling scheme than in the geometric one, there is a larger probability of taking downhill moves under arithmetic scheme (given the same k_B value); on the other hand, the Boltzmann constant controls the value scale for the temperature thus also affecting the probability of taking downhill (small k_B, small downhill probability). Combining these two factors, we are able to achieve the same optimal performance by either using the arithmetic scheme with small k_B, or the geometric scheme with large k_B.

Table 6.3: The best-found meta-parameter settings for Simulated Annealing. Bolded text represents the best setting(s) we found.

MPS^* / Problems	40-site Canola	40-site Wheat	100-site Wheat
Arithmetic Cooling	$k_B^* = 5 \times 10^{-3}$	$k_B^* = 5 \times 10^{-3}$	$k_B^* = 5 \times 10^{-3}$
Geometric Cooling	$k_B^* = 5 \times 10^{-2}$	$k_B^* = 5 \times 10^{-3}$	$k_B^* = 5 \times 10^{-2}$
Step Size Δa	$[\mathbf{0.001},\ 0.005]$	$[\mathbf{0.001},\ 0.005]$	$[\mathbf{0.001},\ 0.005]$
Step Size Δv	$[\mathbf{5},\ 10]$	$[\mathbf{5},\ 10]$	$[\mathbf{5},\ 10]$

We recorded the best-found meta-parameter settings in Table 6.3. To conclude, we found an optimal landscape granularity setting near $\Delta a = 0.001$ and $\Delta v = 5$. Optimal downhill move scheme is found by either using arithmetic cooling with $k_B = 0.005$ or

---

[2] Another assumption is that there exists an optimal scheme of changing the downhill probability over runtime that allow the algorithm to achieve an optimal performance; however, this factor is difficult to measure.

geometric cooling with $k_B = 0.05$. It is generally observed from Table 6.3 that the optimal meta-parameter setting does not vary across different problem instances.[3] Although intuitively, a large problem instance (e.g., 100-site Wheat) usually requires large search steps (e.g., Δa and Δv) in order to maintain an efficient exploration of the larger solution space (e.g., larger delivery rate table $\mathbf{V}$), this effect might be neutralized by an increase in the total number of local search moves used in our experiment for the 100-site problem instance. Similar arguments may be applied to the downhill move scheme, where there is no observed increase of optimal k_B value on the 100-site Wheat instance since the total number of downhill moves also increases with the total number of local search moves. Unless specified explicitly, we will use the default granularity setting at $\Delta a = 0.001$ and $\Delta v = 5$; the downhill move scheme is fixed at arithmetic cooling, and $k_B = 0.005$.

6.2 DNM Local Search

In this experiment, we conducted the tuning of two meta-parameters in DNM local search: the augmentation size δ and the micro-tuning size α. We examined a number of (α, δ) pair settings for each of the studied problem instances and the empirical results are summarized in Table 6.4.

First, Table 6.4 shows that a series of good value settings are found along the row at $\alpha = 0.005$. This indicates that the optimal negative flow α might be independent of the flow augmentation size δ. One feasible explanation is as follows: in DNM local search, the micro-tuning operation reduces delivery rates at farm sites where the overdose effect happens. It is possible that the average effort for alleviating the overdose effect is constant and related only to the underlying nature of the field data (e.g., nutrient response curves in Chapter 2). Although a larger flow augmentation size δ allows more flow to be added into the network thus potentially increases the effort for micro-tuning, it also causes nutrient overdose to happen much earlier.

Second, the optimal setting for δ is found within value range $[0.01, 0.10]$ for all problem instances. We had to reject the intuition that larger problem size requires larger flow

---

[3] It might also be the case that the changes are too small to be observable from our experiments.

Table 6.4: A parameter tuning matrix for DNM local search. δ is the flow augmentation size. α is the micro-tuning size. Each entry gives the average achieved solution quality (total net profit) under one (α, δ) setting. Bolded text indicates the best value of that column.

(a) 40-site Canola problem

α / δ	0.01	0.05	0.1	0.15	0.2
0.001	6941	6972	6980	6965	6970
0.005	**7225**	**7203**	**7176**	**7148**	**7151**
0.01	7136	7128	7100	7076	7062
0.015	7117	7106	7092	7082	7068
0.02	7101	7089	7078	7059	7050

(b) 40-site Wheat problem

α / δ	0.01	0.05	0.1	0.15	0.2
0.001	2469	2545	2563	2578	**2549**
0.005	2538	**2603**	**2588**	**2694**	2539
0.01	**2551**	2598	2582	2583	2539
0.015	2545	2589	2572	2571	2538
0.2	2543	2580	2565	2558	2520

(c) 100-site Wheat problem

α / δ	0.01	0.05	0.1	0.15	0.2
0.001	6214	6186	6160	6180	6135
0.005	**6718**	**6706**	**6664**	**6633**	6460
0.01	6494	6566	6569	6556	6423
0.015	6646	6636	6650	6592	6453
0.2	6633	6651	6642	6607	**6461**

augmentation step. One feasible explanation is as follows: since the flow augmentation operation (Algorithm 6, Step 2) increases the nutrient flows for all farm sites at every local search move, an increase in site number (e.g., from 40-site Wheat to 100-site Wheat) does not necessarily increase the average effort required to conduct the flow augmentation.

To further examine how these two meta-parameters affect algorithm performance, we compared the runtime performance of DNM algorithm under a series of (α, δ) settings. The empirical results are shown in Figure 6.7 and Figure 6.8. It is observed from Figure 6.7 that changes in the α setting has the similar effect as what we found in the SA granularity experiments: large micro-tuning size leads to an early convergence of the algorithm towards good solutions but also suffers from an earlier performance stagnation (e.g., Figure 6.7 at $\delta = 0.01$, $\alpha = 0.015$); small micro-tuning size, on the other hand, considerably increases the runtime cost of the algorithm to reach good solutions thus performing significantly poorly within the given runtime upper bound (e.g., Figure 6.7 at $\delta = 0.01$, $\alpha = 0.001$). An optimal micro-tuning size achieves the balance by providing an appropriate granularity level that enables the algorithm to maintain an efficient exploration of the near-overdose solution region in the fertilizer problem. Similar patterns are also observed for augmentation size δ, as shown in Figure 6.8.

Our next experiment compared three types of nutrient selection schemes (Chapter 5.3.1) including the *random nutrient selection*, *linear heuristic selection* and *proportional heuristic selection* scheme. Empirical results are summarized in Table 6.5. It is observed from Table 6.5 that DNM algorithm performance is less sensitive to different nutrient selection schemes, and heuristic schemes have no demonstrated significant superiority over the random scheme.

We summarized the meta-parameter tuning results in Table 6.6. To conclude, we found an optimal augmentation size δ within range $[0.05, 0.1]$ and micro-tuning size α at 0.005. Generally speaking, DNM algorithm performance is very sensitive to granularity settings (α, δ) and less sensitive to nutrient selection schemes. Unless specified explicitly, our experiments will use a random nutrient selection scheme, and (α, δ) setting would be fixed at the best-found values indicated as bolded text in Table 6.6.

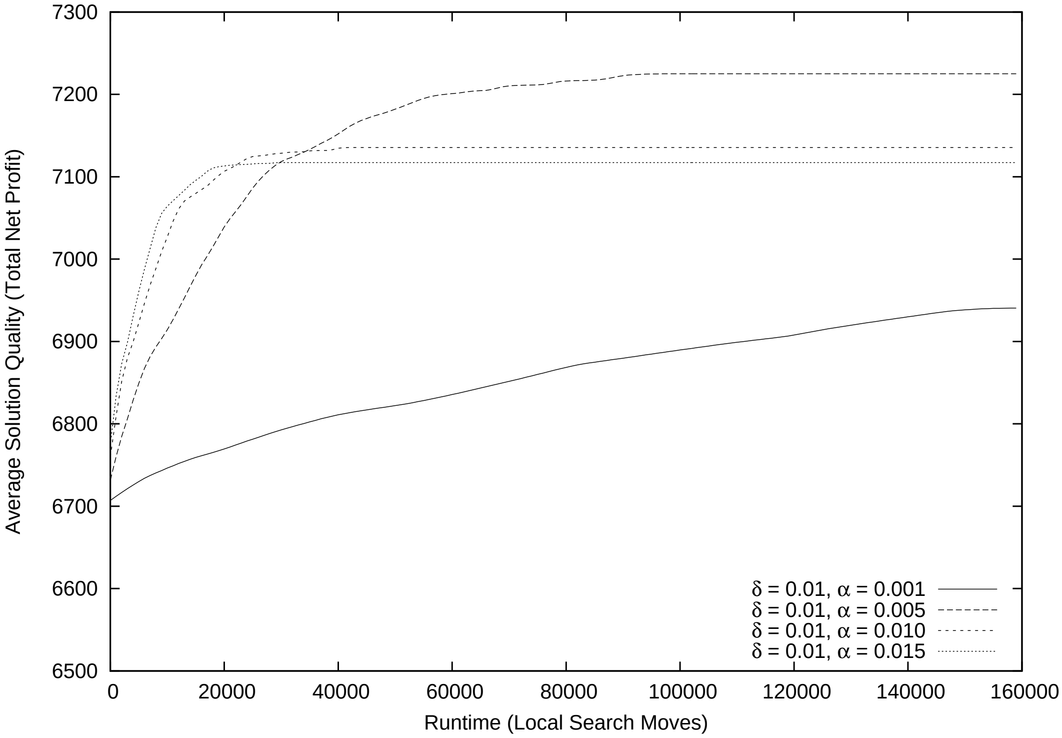

Figure 6.7: Runtime quality comparison of DNM local search under different micro-tuning size α on the 40-site Canola problem.

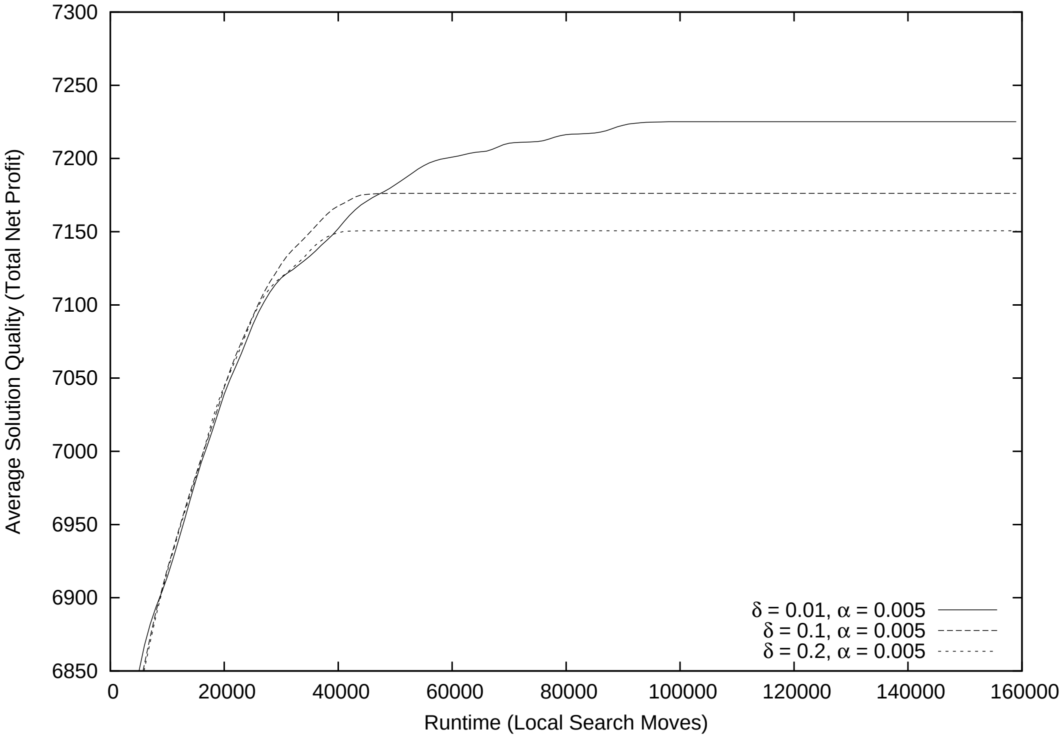

Figure 6.8: Runtime quality comparison of DNM local search under different augmentation step size δ on the 40-site Canola problem.

Table 6.5: Performance comparison of nutrient selection schemes in DNM local search. Each entry gives the average found solution quality under different nutrient selection schemes.

Selection Scheme / Problem	40-site Canola	40-site Wheat	100-site Wheat
Random	7225	2602	**6734**
Linear	**7229**	2610	6690
Proportional	7226	**2616**	6716

Table 6.6: Performance table for DNM local search. Each entry is a text description of the best-found parameter range for the problem. Bolded text indicates the best-found value.

MPS^* / Problem	40-site Canola	40-site Wheat	100-site Wheat
Flow augmentation size δ	$[\mathbf{0.01}, 0.1]$	$[\mathbf{0.05}, 0.15]$	$[\mathbf{0.01}, 0.1]$
Micro-tuning size α	$[\mathbf{0.005}, 0.01]$	$[\mathbf{0.005}, 0.01]$	$\mathbf{0.005}$
Nutrient selection scheme	Random	Random	Random

6.3 Constructive Local Search

6.3.1 RCL Selection Parameter α

Our first experiment evaluated how the RCL parameter α (Chapter 6.4.2) affects the overall performance of the GRASP algorithm. In the experiment, we tested a series of α values and measured $GRASP_{IIA}$ performance under each α setting. The algorithm was run against all three problem instances, and the empirical results are summarized in Table 6.7. Here, we measured a number of performance metrics including: average and best local search runtime to reach 98% of best-known solution quality; average and best-found solution quality for both constructed and final solutions. Since IIA is used as the default local search phase in GRASP, the corresponding standalone IIA performance is also given in Table 6.7 for comparison.

Our first observation is that $GRASP_{IIA}$ outperforms IIA on most measurement met-

Table 6.7: Selection parameter α tuning in $GRASP_{IIA}$. Measurements include: average and best runtime (local search steps) to 98% of best-known solution quality (Avg.RT and Best.RT); average and best-found final solution quality (AvgF.Q and BestF.Q); average and best-found constructed solution quality (AvgC.Q and BestC.Q).

(a) 40-site Canola problem

α	Avg. RT	Best.RT	AvgF.Q	BestF.Q	AvgC.Q	BestC.Q
0(greedy)	10798	9885	**7225**	7292	**3919**	3919
0.2	9085	6541	7214	7283	3374	4356
0.4	7515	3808	7215	7279	3194	5218
0.6	**7125**	**3548**	7211	7294	2570	**6201**
0.8	8763	4430	7219	7288	694	5783
1(random)	9205	4121	7216	**7296**	1247	5428
IIA	*8760*	*3251*	*7217*	*7293*	*	*

(b) 40-site Wheat problem

α	Avg.RT	Best.RT	AvgF.Q	BestF.Q	AvgC.Q	BestC.Q
0(greedy)	94229	14488	2729	2767	**703**	703
0.2	73372	9105	2731	2773	401	1916
0.4	94346	6654	2723	**2779**	365	1739
0.6	57925	**4982**	2733	2766	357	**2042**
0.8	42999	6048	2741	2773	-98	1692
1(random)	**41615**	7000	**2742**	2774	-379	1700
IIA	*51710*	*5549*	*2731*	*2767*	*	*

(c) 100-site Wheat problem

α	Avg.RT	Best.RT	AvgF.Q	BestF.Q	AvgC.Q	BestC.Q
0(greedy)	101350	23367	6964	7014	**1846**	1846
0.2	78410	16917	6984	**7039**	1038	3892
0.4	76057	16933	6980	7025	1171	**5263**
0.6	72937	**14862**	6977	7020	654	4678
0.8	65289	18384	6983	7027	-250	4361
1(random)	**59227**	18636	**6985**	7034	-798	3969
IIA	*93102*	*17075*	*6971*	*7028*	*	*

rics, however, there is no specific α value range that has demonstrated significant superiority on all measurement metrics. Generally speaking, a greedy construction process ($\alpha = 0$) usually leads to worse performance than other schemes; random construction scheme[4] ($\alpha = 1$) has an observed good performance in measurements that compute averaged values (e.g., average runtime and average quality); greedy randomized schemes ($\alpha \in [0.2,\ 0.8]$) usually has a good record in measurements computing the best-found features (e.g., best runtime and best solution quality).

6.3.2 Structure Similarity

Our next experiment analyzed the pairwise structure similarity within both the constructed solution set and the final solution set. The similarity experiment gives an estimation of how solutions obtained by $GRASP_{IIA}$ are "structurally" close to each other in the solution space. For a given solution point set $\mathbf{D}$, assuming two solutions

$$s_1 = (x_1, x_2, ..., x_n),\ s_1 \in \mathbf{D}$$

and

$$s_2 = (y_1, y_2, ..., y_n),\ s_2 \in \mathbf{D}$$

, where x_i and y_i refer to solution variables (components) in the fertilizer problem (either $a_{il} \in \mathbf{A}$ or $v_{lj} \in \mathbf{V}$), the structure similarity between two solution points s_1 and s_2 is defined to be the number of similar components[5] shared between them, in the following form:

$$g(s_1, s_2) = |\{(x_i, y_i)|x_i = y_i,\ x_i \in s_1,\ y_i \in s_2,\ i = 1, 2, ..., n\}|$$

Here, n is the total number of solution variables. Since solution s consists of table $\mathbf{A}$ and table $\mathbf{V}$, the total number of solution variables n is equal to $|\mathbf{A}| + |\mathbf{V}|$. The structure similarity SS for data set $\mathbf{D}$ is calculated in the following way:

$$SS = \sum_{i<j} \frac{g(s_i, s_j)}{K}$$

---

[4] A random construction scheme in GRASP is different from a random start scheme in IIA local search since the granularity setting in GRASP construction leads to a "region sampling" process.

[5] Similarity is defined as equal value assignments between x_i and y_i. However, because of the issue of discretization, equality here refers to similar values within a given value range determined by the granularity level. Here, we use the same granularity setting as described for GRASP construction.

Here K is the total number of solution pairs in data set $\mathbf{D}$. The above formula gives the expected number of similar components that are shared by solution pairs in data set $\mathbf{D}$. In the fertilizer problem, since a solution consists of both mixture table $\mathbf{A}$ and delivery rate table $\mathbf{V}$, we computed the similarity for $\mathbf{A}$ and $\mathbf{V}$ separately. We conducted the similarity test on both the constructed solution points set (generated by the GRASP construction) and the final solutions set obtained by the following IIA local search. For each α value, 100 $GRASP_{IIA}$ independent runs were performed, and 100 constructed and final solution points were obtained. The empirical results are shown in Table 6.8. For comparative purposes, the standalone IIA local search results are also summarized in Table 6.8.

First, it is observed in Table 6.8 that at $\alpha = 0$, the solution set obtained by local search has smaller structure similarity (Final.A and Final.V in Table 6.8) than the constructed set (Cons.A and Cons.V in Table 6.8). At $\alpha = 0$, the construction phase in GRASP becomes a greedy procedure where the same components are promoted and built into the initial solution. However, initializing from the same solutions, different runs of the subsequent local search may reach different local optima. This usually results in a smaller structure similarity to be observed in final solution set. As α increases and more randomness is used, the construction phase allows more components to be added in and usually leads to smaller similarity for both the constructed and final solution set.

Second, the final solution set usually has larger similarity value than the constructed set at $\alpha = 1$.[6] This suggests the fact that good solutions in the fertilizer problem share similar structure.[7] Starting from random positions, the IIA local search converges to a number of local optima where a certain degree of common components are shared.

We also examined the average Euclidean distance between each constructed solution point and its corresponding final solution point (local optima obtained by local search). As shown in Table 6.8, the greedy construction process (α=0) leads to larger initial-final Euclidean distance (Eucli.A and Eucli.V in Table 6.8); a random construction scheme (α=1), however, has relatively smaller Euclidean distance between constructed and final solution points.

---

[6] The only exception is Cons.V and Final.V at $\alpha = 1$ in Table 6.8(a).

[7] According to the Proximate Optimality Principle[25], many combinatorial optimization problems contain good solutions having similar structures.

Table 6.8: Structure similarity test. Measurements include: structure similarity of constructed solutions set (Cons.A and Cons.V); structure similarity of final solutions set (Final.A and Final.V); average Euclidean distance between each constructed solution and its corresponding final solution (Eucli.A and Eucli.V). Numbers in the bracket indicate the smallest euclidean distances within the solutions set.

(a) 40-site Canola

α	Cons.A	Final.A	Cons.V	Final.V	Eucli.A	Eucli.V
0(greedy)	8.0	1.2	67.9	28.0	1.10(1.02)	515(453)
0.2	2.0	0.9	32.7	21.2	0.93(0.66)	453(368)
0.4	1.3	1.0	25.6	19.6	0.79(0.39)	454(339)
0.6	0.6	0.8	21.4	18.5	0.67(0.20)	487(300)
0.8	0.5	0.6	20.4	19.2	0.79(0.25)	564(398)
1(random)	0.4	0.6	20.0	18.5	0.67(0.29)	542(336)
IIA	0.4	0.6	9.7	15.4	0.61(0.12)	491(356)

(b) 40-site Wheat

α	Cons.A	Final.A	Cons.V	Final.V	Eucli.A	Eucli.V
0(greedy)	6.0	1.4	78.0	44.9	1.16(1.12)	664(618)
0.2	1.2	0.8	48.6	23.0	0.97(0.30)	571(335)
0.4	0.6	0.9	30.7	20.9	0.77(0.37)	508(360)
0.6	0.4	0.8	21.8	21.3	0.61(0.22)	513(336)
0.8	0.4	0.9	20.1	22.7	0.66(0.24)	551(421)
1(random)	0.4	0.9	20.0	23.1	0.69(0.15)	575(440)
IIA	0.5	0.9	9.8	20.1	0.63(0.21)	516(390)

(c) 100-site Wheat

α	Cons.A	Final.A	Cons.V	Final.V	Eucli.A	Eucli.V
0(greedy)	6.1	1.9	193.2	111.4	1.21(1.18)	1020(970)
0.2	1.3	1.1	132.7	73.3	1.06(0.71)	913(561)
0.4	0.6	1.2	77.0	57.0	0.87(0.25)	785(588)
0.6	0.3	1.4	52.8	52.7	0.72(0.29)	786(643)
0.8	0.4	1.5	50.4	55.0	0.72(0.19)	830(642)
1(random)	0.4	1.5	50.0	56.1	0.78(0.23)	857(647)
IIA	0.5	1.2	24.4	50.0	0.73(0.25)	786(643)

To conclude, we found that adding construction process to local search (IIA) generally produce better solution quality and improves algorithm performance. We had to reject the intuition that there exists an optimal value range for selection parameter α that achieves an optimal algorithm performance. A greedy randomized construction that combines both greediness and randomness can lead to the finding of better solution points, however, it does not necessarily result in an improvement in average algorithm performance. Unless specified explicitly, our remaining experiments will use $\alpha = 1$ as the default $GRASP_{IIA}$ setting.

6.4 Performance Comparison

In this section, we compared the runtime performance of four studied metaheuristics: *Simulated Annealing (SA)*, *DNM Local Search (DNM)*, *Iterative Improvement Algorithm (IIA)* and *Constructive Local Search ($GRASP_{IIA}$)*. We fixed the meta-parameter of each algorithm at the best-found parameter settings as described in our previous tuning experiments. The algorithms were applied to all three problem instances, and the comparative results are summarized in Figure 6.9, Figure 6.10, Figure 6.11 (Runtime Quality comparison) and Figure 6.12, Figure 6.13, Figure 6.14 (Quality Runtime Comparison).

First, it is observed from Figure 6.9, Figure 6.10 and Figure 6.11 that SA outperforms other algorithms over all problem instances. IIA has been observed to converge more quickly than SA: IIA reaches higher solution quality early in the search. Both SA and $GRASP_{IIA}$ outperform IIA given longer runtime and generally achieves higher solution quality. DNM performs well on the 40-site Canola problem but poorly on the 40-site Wheat and 100-site Wheat problem.

Second, Figure 6.12, Figure 6.13, Figure 6.14 shows that the average runtime cost of our studied algorithms increases significantly on certain quality thresholds (e.g., 98% of best-known quality for Canola problem and 97% for Wheat problem). SA has a smaller runtime cost growth rate compared to both IIA and $GRASP_{IIA}$, and requires less runtime to reach higher solution quality.

To conclude, we found Simulated Annealing (SA) to be an efficient algorithmic so-

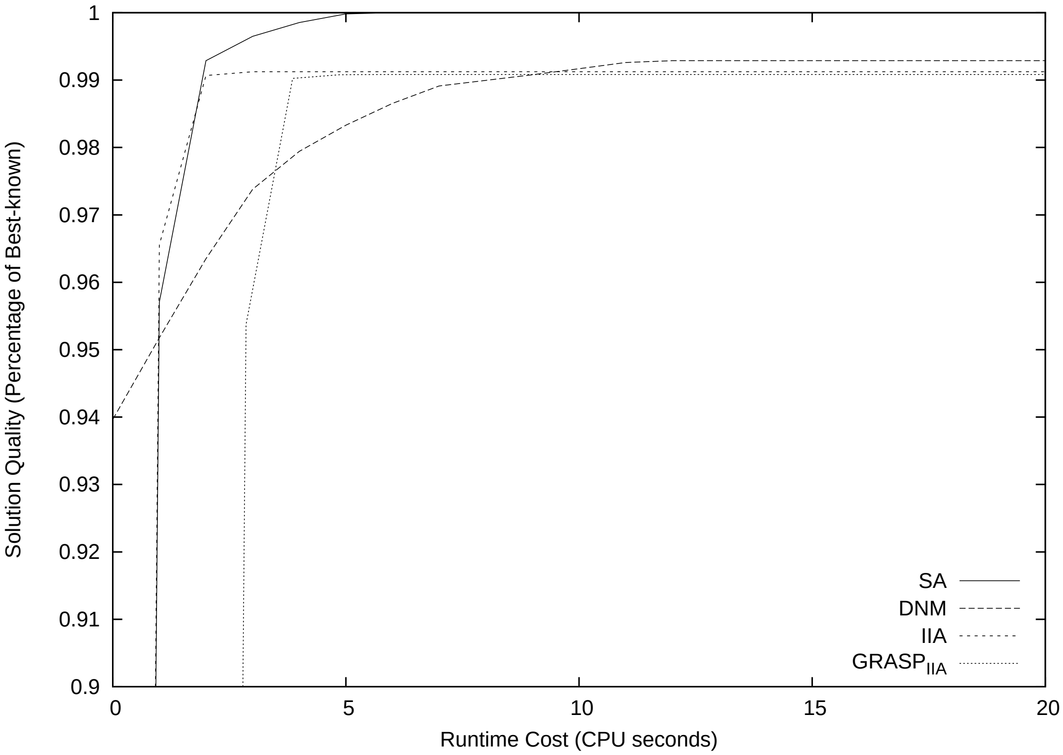

Figure 6.9: Runtime quality comparison on the 40-site Canola problem of four metaheuristics including *Simulated Annealing* (SA), *DNM local search* (DNM), *Iterative Improvement Algorithm* (IIA) and $GRASP_{IIA}$. The figure shows how the best-found solution quality (total net profit averaged over 100 runs) of the algorithm increases over runtime (best-known quality q = 7280).

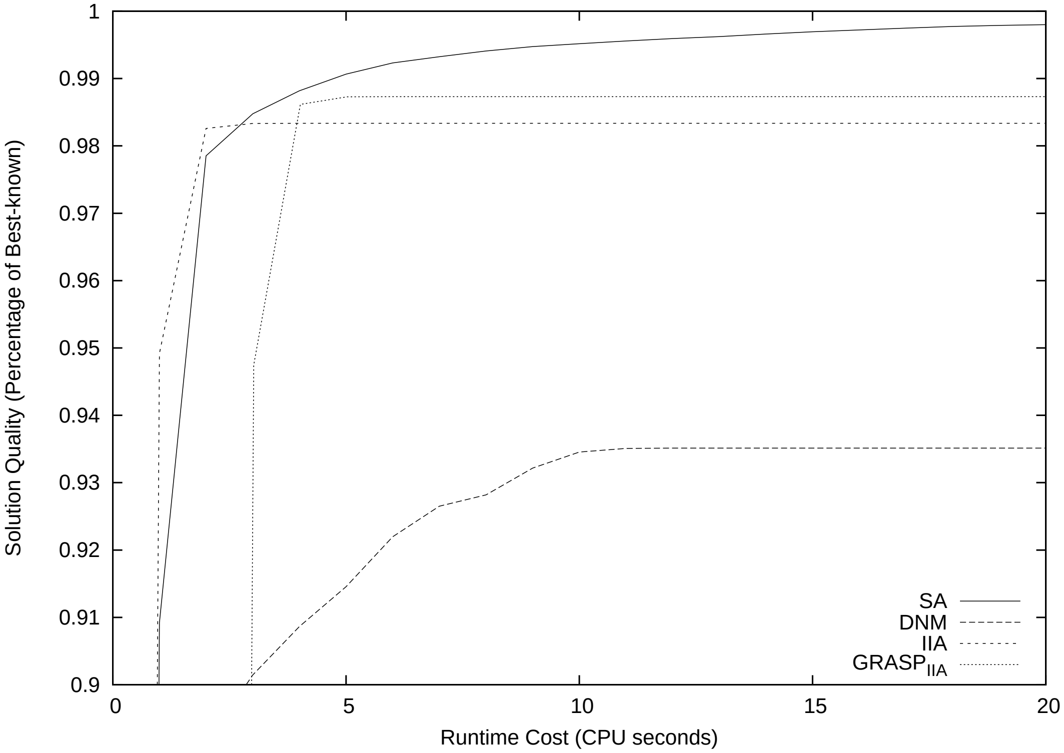

Figure 6.10: Runtime quality comparison on the 40-site Wheat problem of four metaheuristics including *Simulated Annealing* (SA), *DNM local search* (DNM), *Iterative Improvement Algorithm* (IIA) and $GRASP_{IIA}$. The figure shows how the best-found solution quality (total net profit averaged over 100 runs) of the algorithm increases over runtime (best-known quality $q = 2780$).

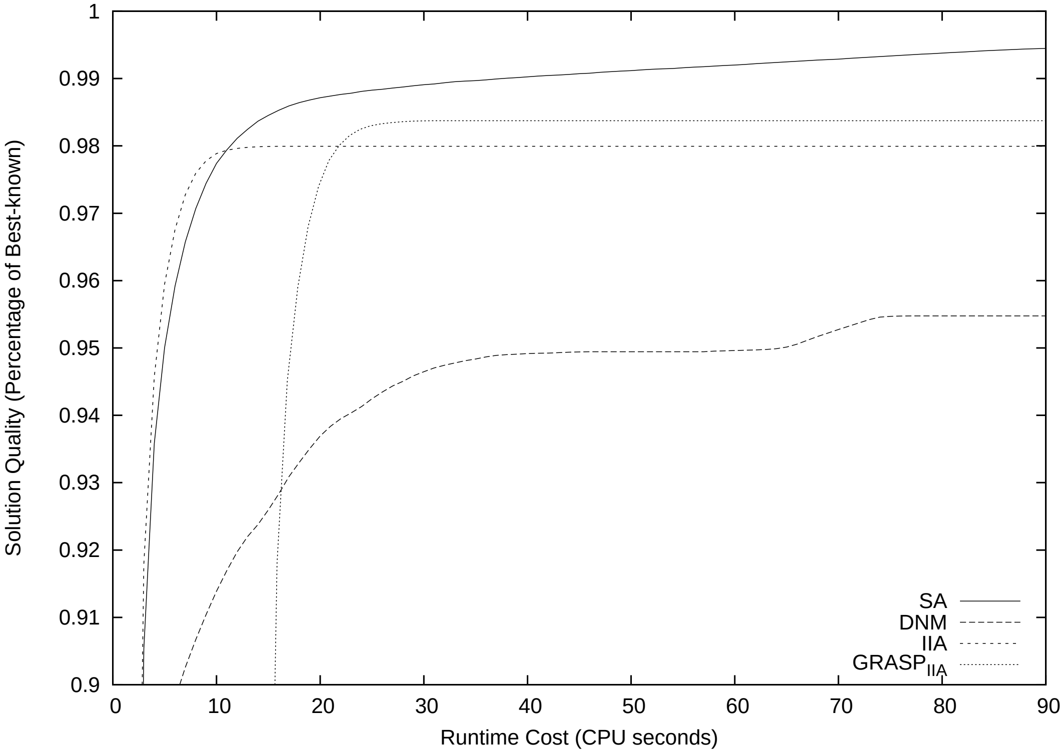

Figure 6.11: Runtime quality comparison on the 100-site Wheat problem of four metaheuristics including *Simulated Annealing* (SA), *DNM local search* (DNM), *Iterative Improvement Algorithm* (IIA) and $GRASP_{IIA}$. The figure shows how the best-found solution quality (total net profit averaged over 100 runs) of the algorithm increases over runtime (best-known quality q = 7100).

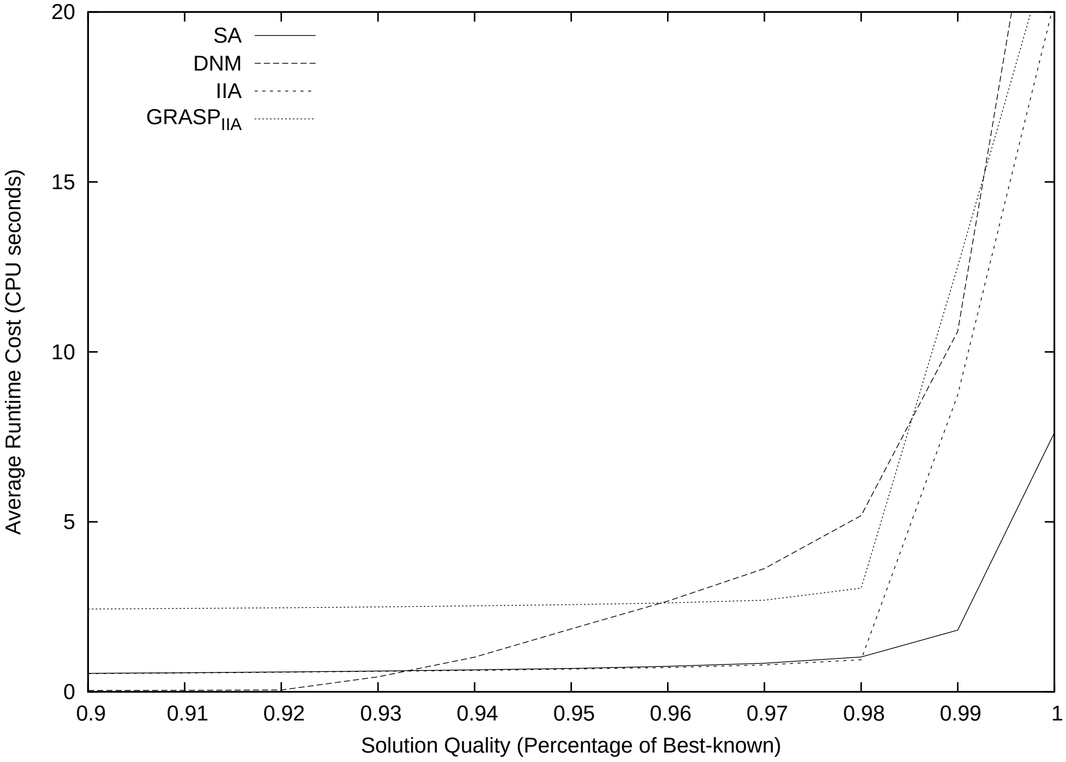

Figure 6.12: Quality runtime measurement on the 40-site Canola problem of four metaheuristics including *Simulated Annealing* (SA), *DNM local search* (DNM), *Iterative Improvement Algorithm* (IIA) and $GRASP_{IIA}$. The figure gives the average runtime cost of the algorithm to reach different solution quality levels (total net profit) for the problem (best-known quality q = 7280).

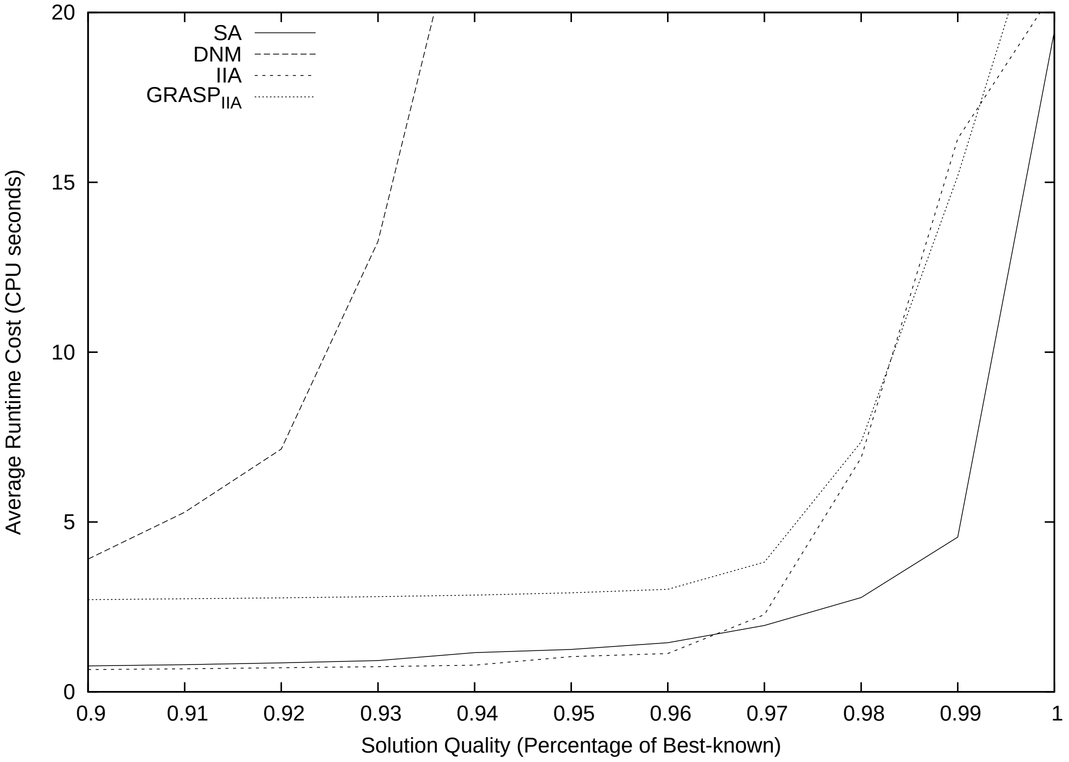

Figure 6.13: Quality runtime measurement on the 40-site Wheat problem of four metaheuristics including *Simulated Annealing* (SA), *DNM local search* (DNM), *Iterative Improvement Algorithm* (IIA) and $GRASP_{IIA}$. The figure gives the average runtime cost of the algorithm to reach different solution quality levels (total net profit) for the problem (best-known quality $q = 2780$).

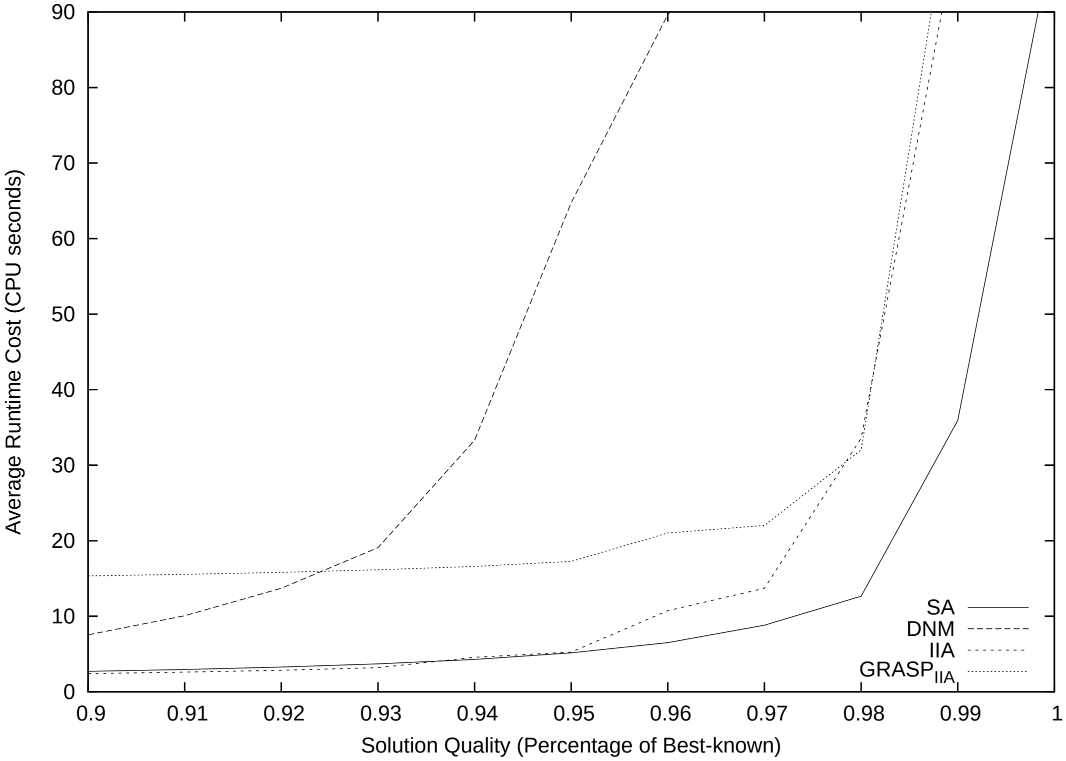

Figure 6.14: Quality runtime measurement on the 100-site Wheat problem of four meta-heuristics including *Simulated Annealing* (SA), *DNM local search* (DNM), *Iterative Improvement Algorithm* (IIA) and $GRASP_{IIA}$. The figure gives the average runtime cost of the algorithm to reach different solution quality levels (total net profit) for the problem (best-known quality q = 7100).

lution for the fertilizer problem, and the Iterative Improvement Algorithm (IIA) also performs surprisingly well. The Constructive Local Search ($GRASP_{IIA}$) produces better solutions than IIA but also brings in extra computing cost for initial solution construction.

6.5 One Compound Blend vs. Two Compound Blend

We investigate in this section whether there is an advantage of using two compound blends instead of one. In the case of one compound blend, both mixture table $\mathbf{A}$ and delivery rate table $\mathbf{V}$ contain one column. We applied all four studied algorithms to the fertilizer problem using one compound blend and measured both the average achieved yield and total net profit. The comparative results are summarized in Table 6.9.

Empirical results show that, using two blends instead of one, there is an on-average 3% to 5% increase in yield[8] on the 40-site Canola problem, and 9% to 12% yield increase on the 40-site Wheat and 100-site Wheat problem.

Current farm machinery has limits on using only two compound blends, however, it is conceivable to deliver more than two blends. Our empirical results show that using two compound blends generally improves the solution quality, compared to the one blend situation. However, further adding of the compound blend (e.g., three or four blends) might have a reduced rate of improvement in solution quality, since using two compound blends has provided a reasonably large solution space for including many good nutrient delivery schemes, and it is not clear how much space there is for further improvement[9]. It is also not clear whether the improvement of using more blends will be noticeable given the uncertainty of other factors during farming (e.g., weather condition).

6.6 Hybrid Metaheuristics Using GRASP

We also investigated the possibility of hybridizing other local search metaheuristics into GRASP, including DNM local search and Simulated Annealing. Our experiments show

---

[8] We measure the improvement in yield here since the crop market value L is subject to change each year. A default value of $L = 3.5$ (dollars per bushel) is used in our experiment for calculating the net profit.

[9] To know this, further experimental work is needed.

Table 6.9: Comparison of algorithm performance using one compound blend and two compound blends. Measurements include: average achieved yield using one blend (Yield.1) and two blends (Yield.2), in bushels; average delivery cost under one blend (Cost.1) and two blends (Cost.2), in dollars; average achieved total net profit using one blend (Profit.1) and two blends (Profit.2), in dollars.

(a) 40-site Canola, maximum total yield $Y_{max} = 2863$

Algs	Yield.1	Yield.2	Yield Increase	Cost.1	Cost.2	Profit.1	Profit.2
SA	2259	2422	5.7%	922	1162	6984	7314
DNM	2295	2394	3.5%	1145	1151	6889	7228
IIA	2252	2383	4.6%	908	1124	6975	7217
$GRASP_{IIA}$	2289	2388	3.5%	962	1143	7048	7214

(b) 40-site Wheat, maximum total yield $Y_{max} = 1452$

Algs	Yield.1	Yield.2	Yield Increase	Cost.1	Cost.2	Profit.1	Profit.2
SA	1029	1212	12.6%	1164	1464	2439	2777
DNM	1016	1175	11.0%	1293	1510	2264	2602
IIA	1020	1190	11.7%	1147	1434	2422	2731
$GRASP_{IIA}$	1057	1199	9.8%	1198	1453	2501	2743

(c) 100-site Wheat, maximum total yield $Y_{max} = 3605$

Algs	Yield.1	Yield.2	Yield Increase	Cost.1	Cost.2	Profit.1	Profit.2
SA	2580	3059	13.3%	2881	3639	6149	7066
DNM	2670	2948	7.7%	3365	3582	5982	6738
IIA	2583	3021	12.1%	2872	3603	6168	6971
$GRASP_{IIA}$	2680	3034	9.8%	3009	3630	6370	6987

that $GRASP_{DNM}$ performs significantly poorly on all three problem instances. A detailed look at the experimental data shows that the $\mathbf{A}$ table and $\mathbf{V}$ table generated by the GRASP construction process contain entries of large value (either large a_{il} in $\mathbf{A}$, or high delivery rate v_{lj} in $\mathbf{V}$). Since the greedy construction phase in GRASP promotes solution elements having larger contribution to the partial solution quality,[10] this usually leads to a number of high ranking elements being selected into the constructed solution $\mathbf{s_0}$. In the subsequent DNM local search phase, the flow augmentation further adds nutrient flows into farm sites having high delivery rate, causing nutrient overdose rather early. Although the DNM micro-tuning process attempts to alleviate this effect by subtracting flows from the network, the significant effort required to do so usually leads to an observed poor performance. In terms of the search landscape in DNM, this indicates the situation where the DNM local search is trapped in local optima and fails to move away from the initial attraction basin(s) identified by the GRASP construction.

The experiments for $GRASP_{SA}$, however, shows the hybridization to be rather unstable: small changes in a SA meta-parameter setting (e.g., the downhill scheme) might result in different algorithm performance to be observed. Generally speaking, SA local search is less sensitive to initial search positions in our studied fertilizer problem, mainly due to the use of downhill moves. In the SA search landscape, this indicates the fact that SA is able to move away from the initial basin(s) identified by the GRASP construction phase and reaches other regions of the solution space. This makes our performance evaluation of $GRASP_{SA}$ less valuable in terms of algorithm comparison, since the observed performance of $GRASP_{SA}$ is very similar to what has been observed in SA local search experiments. Furthermore, the computational cost for constructing an initial solution is high, however, the constructed initial solution does not affect the final solution quality obtained by SA very much.

---

[10] This is due to the greedy function we defined: g(e) = f($s_0 \cup \{$e$\}$) - f(s_0)

CHAPTER 7

CONCLUSION AND FUTURE WORK

7.1 Conclusion and Discussion

In this work, we conducted an empirical study of several state-of-the-art metaheuristic algorithms and applied them to the fertilizer optimization problem. We found that Simulated Annealing (SA) is an efficient algorithmic solution to the problem and outperforms other metaheuristics. Downhill moves, although bringing in extra computing cost, allow the local search to explore larger regions of the solution space and generally result in higher solution quality.

Second, we found that the Iterated Improvement Algorithm (IIA) performs surprisingly well, given the fact that there exists a large number of good local optima in the fertilizer problem search landscape, and these local optima regions (basins of attraction) are highly overlapped.[1] The use of a random neighborhood selection in IIA enables a strict uphill search to traverse multiple attraction basins before a local optimum is reached. Generally speaking, this leads to the finding of good solutions.

We also proposed and implemented in this work a problem-specific algorithm to the fertilizer problem: DNM local search. Empirical results show that DNM performs well on the Canola instance but poorly on the Wheat problem instances. Using nutrient delivery network is an efficient way of exploring the high-dimensional solution space of the fertilizer problem, however, substantial experimental tuning effort is required to find the balance between the flow augmentation size and the micro-tuning size that achieves a good algorithm performance.

---

[1] The overlap feature can be inferred from the fact that in the fertilizer problem, starting from one initial search position, different independent runs of IIA local search reaches different local optima.

Our study of the Constructive Local Search method shows that combining construction process with local search ($GRASP_{IIA}$) can lead to the finding of better solutions. However, the improvement in overall algorithm performance is not significant, and there is no specific α value range that achieves an optimal algorithm performance. The greedy randomized construction process might produce initial solutions having high quality level. However, applying the subsequent local search on these good initial points does not necessarily leads to the finding of high quality local optima: our empirical experiments show that there is no observed strong correlation between constructed solution quality and final solution (local optima) quality in the fertilizer problem. It is also surprising to find that a random construction process ($\alpha = 1$) yields good results: independent runs of the local search (IIA) are initialized at different regions of the solution space thus providing a uniform re-distribution of search effort within the high dimensional solution space. However, it is generally observed from our experiments that IIA local search performance is less sensitive to initial search positions in the context of high dimensional search landscape of our studied fertilizer optimization problem.

We also demonstrated in this work how the experimental tuning work can be applied to obtain a good performance of our studied metaheuristic algorithms. The *local search granularity* (move step sizes) and the *downhill move scheme* are two major types of meta-parameters that affect the overall performance of the algorithms. An optimal granularity setting (e.g., step size Δa, Δv in SA and IIA, flow augmentation size δ and micro-tuning size α in DNM) achieves the balance between search efficiency and landscape "accuracy" by discretizing the continuous solution space at an appropriate granularity level and including a large number of good discrete solution points. Our empirical experiments show that we are able to approximate the optimal granularity through a combinatorial examination of several associated parameters together (e.g., a granularity tuning matrix in both SA and DNM experiments). However, significant experimental time is usually required.

The optimal downhill move scheme (e.g., cooling schedule and k_B in SA), on the other hand, achieves balance between search diversification and intensification. Our empirical results show that adding downhill move to strict uphill search (e.g., IIA as an extreme case of intensified search) improves algorithms' performance by allowing the search to conduct a more diversified exploration of the solution space; furthermore, our tuning result in Simulated Annealing (SA) suggests that there seems to exist an "optimal" distribution of downhill moves over runtime that allows SA to achieve an optimal algorithm performance using different parameter settings (e.g., arithmetic cooling with small k_B, or geometric cooling with large k_B).

7.2 Future Work

The significant effort required for metaheuristics performance tuning remains a major challenge. Although our empirical results show that the optimal meta-parameter setting does not vary significantly across fertilizer problem instances, the finding of such optimal setting usually requires a combinatorial examination of various parameters. There has been an increasing interest in the literature applying machine learning techniques to automate the meta-parameter tuning process [40, 39]. The objective is to achieve an on-line prediction of the optimal meta-parameter settings for each encountered problem instance so that more robust algorithmic solutions can be produced.

Our study in this work assumed that the objective function is a black box. Future work may be extended into exploring certain features of the profit function that might provide intuitions for new search algorithm design. Our structure similarity test in $GRASP_{IIA}$ experiment indicates that good solutions in the fertilizer optimization problem share similar structures. One intuition is to conduct a post-analysis of the set of good solutions for the problem and construct probabilistic models to capture the structure similarity of the good solution set [56, 58, 57, 59]. Integrating a probability model into metaheuristic algorithm framework might help us predict and identify good solution regions, leading to an improvement in overall search efficiency.

Throughout our experiments, a number of performance metrics have been adopted including RunTime Distributions (RTD), Runtime Quality and Quality Runtime. RTD measurement is sensitive to algorithm performance since it computes the frequency with which the algorithm reaches a pre-defined solution quality threshold (e.g., 98$\%$ of the reference solution quality) within multiple runs. Our experiment has been using the best known solution quality as the reference solution quality, since we cannot find the global optima using currently implemented techniques. To improve the performance evaluation work, it is beneficial to obtain the global optima using global optimization packages in some existing mathematical tools, for example, GAMS [18] and Maple [52].

The scope of our work is limited to solving combinatorial (discrete) optimization problems by searching. Metaheuristic algorithms (an alternative name is Stochastic Local Search) belongs to stochastic search techniques where randomness is used and non-deterministic factors are applied.[2] We aim at providing acceptable good solutions within limited runtime and assume no utilization of the target problem structure (black box optimization). Future work could be extended into investigating various global optimization techniques for hard combinatorial optimization problem solving. Our intuition is that this requires effort investigating the target problem structure (e.g., net profit function in the fertilizer problem) using analytical methods in order to provide feasible algorithmic solutions.

---

[2] Its counterpart is deterministic algorithms such as branch-and-bound that conducts a complete (exhaustive) investigation of the problem solution space.

REFERENCES

[1] *SEMINAL Network: Software Engineering with Metaheuristic INnovative ALgorithms*. http://www.brunel.ac.uk/ csstmmh2/seminal/.

[2] Aarts, E., and Laarhoven, P. Statistical Cooling: A General Approach to Combinatorial Optimization. *Phillips Journal of Research 40* (1985), 193–226.

[3] Ahuja, R. K., Magnanti, T. L., and Orlin, J. B. *Network Flows: Theory, Algorithms and Applications*. Prentice Hall, 1993.

[4] Ball, M. O., Magnanti, T. L., Monma, C. L., and Nemhauser, G. L., Eds. *Network Routing*, vol. 8 of *Handbooks In Operations Research And Management Science*. 1995.

[5] Blum, C., and Roli, A. Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison. *ACM Computing Surveys 35*, 3 (September 2003), 268–308.

[6] Boender, C. G. E., and Romeijn, H. E. Stochastic Methods. In *Handbook of Global Optimization*, R. Horst and P. M. Pardalos, Eds., vol. 2 of *Nonconvex Optimization and its Applications*. Kluwer Academic Publishers, 1995.

[7] Chiarandini, M., and Stützle, T. A Landscape Analysis for A Hybrid Algorithm on Timetabling Problem. Technical Report No. AIDA-03-05, FG Intellektik, FB Informatik, TU Darmstadt, Germany, April 2003.

[8] Coello, C. A., and Cortés, N. C. Solving Multiobjective Optimization Problems using an Artificial Immune System. *Genetic Programming and Evolvable Machines 6*, 2 (June 2005).

[9] Díaz, B. D. *The VRP Web*. AUREN and the Languages and Computation Sciences Department of the University of Málaga, http://neo.lcc.uma.es/radi-aeb/WebVRP/, 2005.

[10] Díaz, D. M. +CARPS: Configuration of Metaheuristics Based on Cooperative Agents. In *HM '04: Proceedings of the First International Workshop On Hybrid Metaheuristics* (August 2004), pp. 115–126.

[11] Dixon, L. C. W. Global optima without convexity. Tech. rep., Numerical Optimization Centre, Hatfield Polytechnic, Hatfield, England, 1978.

[12] DORIGO, M., AND CARO, G. D. The Ant Colony Optimization Meta-Heuristic. In *New Ideas in Optimization*, D. Corne, M. Dorigo, and F. Glover, Eds. McGraw-Hill, London, UK, 1999, pp. 11–32.

[13] DORIGO, M., MANIEZZO, V., AND COLORNI, A. The Ant System: Optimization by a Colony of Cooperating Agents. *IEEE Transactions on System, Man and Cybernetics-Part B 26*, 1 (1996), 1–13.

[14] DORNE, R., AND VOUDOURIS, C. HSF: A Generic Framework to Easily Design Meta-Heuristic Methods. In *Metaheuristics: Computer Decision-Making*, M. G. Resende and J. P. de Sousa, Eds., vol. 86 of *Applied Optimization*. Springer, 2004.

[15] FINK, A., AND VOSS, S. Reusable Metaheuristic Software Components and their Application via Software Generators. In *Metaheuristics: Computer Decision-Making*, M. G. Resende and J. P. de Sousa, Eds., vol. 86 of *Applied Optimization*. Springer, 2004.

[16] FONLUPT, C., ROBILIARD, D., PREUX, P., AND TALBI, E. G. Fitness Landscapes and Performance of Meta-heuristics. In *Metaheuristics – Advances and Trends in Local Search Paradigms for Optimization*. Kluwer Academic Press, 1999, ch. 18, pp. 255–266.

[17] FUDENBERG, D., AND TIROLE, J., Eds. *Game Theory*. 1983.

[18] GAMS DEVELOPMENT CORPORATION. *The General Algebraic Modeling System (GAMS)*. http://www.gams.com.

[19] GAREY, M. R., AND JOHNSON, D. S., Eds. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. 1979.

[20] GARNIER, J., AND KALLEL, L. Efficiency of Local Search with Multiple Local Optima. *SIAM Journal on Discrete Mathematics 15*, 1 (2002), 122–141.

[21] GASPERO, L. D., AND SCHAERF, A. EasyLocal++: An Object-Oriented Framework for the Design of Local Search Algorithms and Metaheuristics. In *Metaheuristics: Computer Decision-Making*, M. G. Resende and J. P. de Sousa, Eds., vol. 86 of *Applied Optimization*. Springer, 2004.

[22] GENDREAU, M. An Introduction to Tabu Search.

[23] GLOVER, F., AND KOCHENBERGER, G. Metaheuristic Agent Processes (MAPs). In *Metaheuristics: Progress as Real Problem Solvers*. Springer, Operation Research and Decision Theory, 2003.

[24] GLOVER, F., AND KOCHENBERGER, G. A. Asynchronous Teams. In *Handbook of Metaheuristics*. Kluwer Academic Publishers, 2003.

[25] GLOVER, F., AND LAGUNA, M. Tabu Search. In *Modern Heuristic Techniques for Combinatorial Problems* (Oxford, England, 1993), C. Reeves, Ed., Blackwell Scientific Publishing.

[26] GLOVER, F., AND LAGUNA, M., Eds. *Tabu Search*. Kluwer Academic Publishers, 1997.

[27] GLOVER, F. W., AND KOCHENBERGER, G. A., Eds. *Handbook of Metaheuristics*. Kluwer's International Series in Operations Research and Management Science. Kluwer Academic Publishers, 2003.

[28] HANSEN, P., AND MLADENOVIĆ, N. Variable Neighborhood Decomposition Search. *Journal of Heuristics 7* (2001), 335–350.

[29] HANSEN, P., AND MLADENOVIĆ, N. A Tutorial on Variable Neighborhood Search. *GERAD, Group for Research in Decision Analysis* (2003).

[30] HANSEN, P., AND MLADENOVIĆ, N. Variable Neighborhood Search. In *Handbook of Metaheuristics*. Kluwer Academic Publishers, 2003.

[31] HARALICK, R., AND ELLIOT, G. Increasing Tree Search Efficiency for Constraint Satisfaction Problems. In *Artificial Intelligence*, vol. 14. 1980, pp. 263–313.

[32] HARMAN, M., AND JONES, B. F. The SEMINAL workshop: Reformulating Software Engineering as a Metaheuristic Search Problem. *SIGSOFT Softw. Eng. Notes 26*, 6 (2001), 62–66.

[33] HART, E., ROSS, P., AND CORNE, D. Evolutionary Scheduling: A Review. *Genetic Programming and Evolvable Machines 6*, 2 (June 2005).

[34] HENDERSON, D., JACOBSON, S. H., AND JOHNSON, A. W. Greedy Randomized Adaptive Search Procedures. In *Handbook of Metaheuristics*. Kluwer Academic Publishers, 2003.

[35] HENDERSON, D., JACOBSON, S. H., AND JOHNSON, A. W. The Theory and Practice of Simulated Annealing. In *Handbook of Metaheuristics*. Kluwer Academic Publishers, 2003.

[36] HOGG, T., AND HUBERMAN, B. A. Better Than The Best: The Power of Cooperation. In *Lectures in Complex Systems*. Addison-Wesley, 1993, pp. 165–184.

[37] HOOS, H. H., AND STÜTZLE, T. *Stochastic Local Search: Foundations and Applications*. Morgan Kaufmann, 2005.

[38] HU, X. *Particle Swarm Optimization tutorial*. www.swarmintelligence.org, 2003.

[39] HUTTER, F., AND HAMADI, Y. Parameter Adjustment Based on Performance Prediction: Towards an Instance-Aware Problem Solver. Tech. Rep. MSR-TR-2005-125, Microsoft Research, Redmond, WA 98052.

[40] HUTTER, F., HAMADI, Y., HOOS, H. H., AND LEYTON-BROWN, K. Performance Prediction and Automated Tuning of Randomized and Parametric Algorithms. In *Twelfth International Conference on Principles and Practice of Constraint Programming (CP06)* (2006).

[41] JONES, T. C. *Evolutionary Algorithms, Fitness Landscapes and Search*. PhD thesis, Univ. of New Mexico, 1995.

[42] JONES, T. C. One Operator, One Landscape. Santa Fe Institute, 1399 Hyde Park Road, Santa Fe, NM 87501, USA, 1995.

[43] K. F. DOERNER AND W. J. GUTJAHR AND R. F. HARTL AND C. STRAUSS AND C. STUMMER. Pareto Ant Colony Optimization With ILP Preprocessing In Multiobjective Project Portfolio Selection. *European Journal of Operational Research* (2005).

[44] KAN, A. H. G. R., AND TIMMER, G. T. Global Optimization. In *Handbooks in Operations Research and Management Science*, vol. 1 of *Optimization*. North-Holland, Amsterdam, 1989, pp. 631–662.

[45] KENNEDY, J., AND EBERHART, R. C. Particle Swarm Optimization. *Proceedings of IEEE International Conference on Neural Networks (Perth, Australia) 4* (1995), 1942–1948.

[46] KIRKPATRICK, S., GELATT, C. D., AND VECCHI, M. P. Optimization by Simulated Annealing. *SCIENCE 220*, 4598 (May 1983), 671–678.

[47] K.MIYAMOTO, AND YASUDA, K. Multipoint-based Tabu Search Using Proximate Optimality Principle. *IEEE International Conference on Systems, Man and Cybernetics 4* (Oct. 2005), 3094 – 3099.

[48] LAU, H. C., AND WANG, H. A Multi-Agent Approach for Solving Optimization Problems involving Expensive Resources. In *SAC '05: Proceedings of the 2005 ACM symposium on Applied computing* (2005), ACM Press, pp. 79–83.

[49] LUNDY, M., AND MEES, A. Convergence of an Annealing Algorithm. *Mathematical Programming 34* (1986), 111–124.

[50] MACKWORTH, A. Consistency in Networks of Relations in Artificial Intelligence. In *Artificial Intelligence*, vol. 8. 1977, pp. 99–118.

[51] MAIER, H. R., SIMPSON, A. R., ZECCHIN, A. C., FOONG, W. K., PHANG, K. Y., SEAH, H. Y., AND TAN, C. L. Ant Colony Optimization for Design of Water Distribution Systems. *Journal of Water Resources Planning And Management 129*, 3 (May/June 2003), 200–209.

[52] MAPLESOFT. *Maple*. http://www.maplesoft.com.

[53] MILANO, M., AND ROLI, A. MAGMA: A Multiagent Architecture for Metaheuristics. *IEEE Transactions On System, Man, And Cybernetics - Part B. Cybernetics 33*, 2 (April 2004).

[54] P. MOSCATO. *TSPBIB*. http://www.ing.unlp.edu.ar/cetad/mos/TSPBIB_home.html.

[55] PARADIMITRIOU, C., AND STEIGLITZ, K., Eds. *Combinatorial Optimization: Algorithms and Complexity*. Dover Publications Inc., New York, 1982.

[56] PELIKAN, M. *Bayesian Optimization Algorithm (BOA)*. http://www.cs.umsl.edu/ pelikan/boa.html, 2005.

[57] PELIKAN, M., GOLDBERG, D. E., AND CANTÚ-PAZ, E. BOA: The Bayesian Optimization Algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference GECCO-99* (Orlando, FL), vol. I, Morgan Kaufmann Publishers, San Fransisco, CA.

[58] PELIKAN, M., GOLDBERG, D. E., AND LOBO, F. A Survey of Optimization by Building and Using Probabilistic Models. *Computational Optimization and Applications 21*, 1 (2002), 5–20.

[59] PELIKAN, M., GOLDBERG, D. E., OCENASEK, J., AND TREBST, S. Robust and Scalable Black-Box Optimization, Hierarchy and Ising Spin Glasses. IlliGAL Report No. 2003019, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign, Urbana, IL, 2003.

[60] PINTÉR, J. D., Ed. *Global Optimization in Action: Continuous and Lipschitz Optimization, Algorithm, Implementation and Applications*, vol. 6 of *Nonconvex Optimization and Its Applications*. Kluwer Academic Publishers, 1996.

[61] PITSOULIS, L. S., AND RESENDE, M. G. C. Greedy Randomized Adaptive Search Procedures. Tech. Rep. TD-53RSJY, AT&T Labs Research, 2001.

[62] PRASAD, N., LANDER, S. E., AND LESSER, V. R. Cooperative Learning over Composite Search Spaces: Experiences with a Multi-agent Design System. *Proceedings of the Thirteenth National Conference on Artificial Intelligence 1* (January 1996), 68–73.

[63] R.E. BURKARD AND S.E. KARISCH AND F. RENDL. *QAPLIB - A Quadratic Assignment Problem Library*, 1997.

[64] REGO, C., AND GLOVER, F. In *The Traveling Salesman Problem and Its Variations*, G. Gutin and A. Punnen, Eds. Kluwer Academic Publishers, Combinatorial Optimization Series, 2002, ch. 12, pp. 309–368.

[65] REINELT, G. TSPLIB - A Traveling Salesman Problem Library. vol. 3, ORSA J. Comput., pp. 376–384.

[66] RUSSELL, S., AND NORVIG, P. *Artificial Intelligence: A Modern Approach*, second ed. Prentice Hall Series in Artificial Intelligence, 2003.

[67] SCHUTTE, J. F., REINBOLT, J. A., FREGLY, B. J., HAFTKA, R. T., AND GEORGE, A. D. Parallel Global Optimization With The Particle Swarm Algorithm. *International Journal of Numerical Methods in Engineering 61* (2004), 2296–2315.

[68] SHI, Y., AND EBERHART, R. C. Empirical Study of Particle Swarm Optimization. *Proceedings of the 1999 Congress on Evolutionary Computation* (1999), 1945–1950.

[69] SHMYGELSKA, A., AND HOOS, H. H. An Improved Ant Colony Optimization Algorithm for the 2D HP Protein Folding Problem. In *Proc. of the Sixteenth Canadian Conference on Artificial Intelligence (AI'2003)* (2003).

[70] STADLER, P. F. Fitness Landscapes. In *Biological Evolution and Statistical Physics*, M. Lässig and A. Valleriani, Eds. Springer-Verlag, Berlin, 2002, pp. 187–207.

[71] VINCENT, P., AND RUBIN, I. A Framework and Analysis for Cooperative Search Using UAV Swarms. In *SAC '04: Proceedings of the 2004 ACM symposium on Applied computing* (2004), ACM Press, pp. 79–86.

[72] WHITLEY, D. *A Genetic Algorithm Tutorial.* Computer Science Department, Colorado State University, http://samizdat.mines.edu/ga_tutorial.

[73] WOLPERT, D. H., AND MACREADY, W. G. No Free Lunch Theorems for Search. Tech. rep., The Santa Fe Institute, Santa Fe, NM, 1995.

[74] WUENSCHE, A. *Attractor Basins Of Discrete Networks - Implications of self-organization and memory*. PhD thesis, Univ. of Sussex, 1997.

[75] WUENSCHE, A. Genomic Regulation Modeled As a Network With Basins of Attraction. *Pac. Symp. Biocomput* (1998), 89–102.

[76] YASUDA, K., AND KANAZAWA, T. Proximate Optimality Principle Based Tabu Search. *IEEE International Conference on Systems, Man and Cybernetics 2* (Oct. 2003), 1560 – 1565.

APPENDIX A

SAMPLE TABLES

An example of the nutrients set $\mathbf{M}$ containing four market nutrients:

$$\mathbf{M} = \begin{pmatrix} \mathbf{m}_1 & \mathbf{m}_2 & \mathbf{m}_3 & \mathbf{m}_4 \end{pmatrix}_{4\times 4} = \begin{pmatrix} 46 & 11 & 0 & 21 \\ 0 & 55 & 0 & 0 \\ 0 & 0 & 60 & 0 \\ 0 & 0 & 0 & 24 \end{pmatrix}_{4\times 4}$$

An example of the nutrient mixture table $\mathbf{A}$ for $\mathbf{M}$:

$$\mathbf{A} = \begin{pmatrix} \mathbf{a}_1 & \mathbf{a}_2 \end{pmatrix}_{4\times 2} = \begin{pmatrix} 0.5 & 0.1 \\ 0.0 & 0.3 \\ 0.2 & 0.0 \\ 0.3 & 0.6 \end{pmatrix}_{4\times 2}$$

An example of the compound blends $\mathbf{B}$:

$$\mathbf{B} = \mathbf{MA} = \begin{pmatrix} \mathbf{b}_1 & \mathbf{b}_2 \end{pmatrix}_{4\times 2} = \begin{pmatrix} 29.3 & 20.5 \\ 0.0 & 16.5 \\ 12.0 & 0.0 \\ 7.2 & 14.4 \end{pmatrix}_{4\times 2}$$

An example of the delivery rate vector $\mathbf{v}_j$ at site j:

$$\mathbf{v}_j = \begin{pmatrix} 137 \\ 82 \end{pmatrix}_{2\times 1}$$

An example of the blend delivery $\mathbf{d}_j$ at site j, in lbs per acre:

$$\mathbf{d}_j = \mathbf{B}\mathbf{v}_j = \begin{pmatrix} 56.95 \\ 13.53 \\ 16.44 \\ 21.67 \end{pmatrix}_{4\times 1}$$