THE HONG KONG POLYTECHNIC UNIVERSITY

DEPARTMENT OF INDUSTRIAL AND SYSTEMS ENGINEERING

A SUBOPTIMUM- AND PROPORTION-BASED HEURISTIC GENERATION METHOD FOR COMBINATORIAL OPTIMIZATION PROBLEMS

Fan XUE

A Thesis Submitted in Partial Fulfilment of
the Requirements for the Degree of
Doctor of Philosophy

September 2012

CERTIFICATE OF ORIGINALITY

I hereby declare that this thesis is my own work and that, to the best of my knowledge and belief, it reproduces no material previously published or written, nor material that has been accepted for the award of any other degree or diploma, except where due acknowledgement has been made in the text.

_____

XUE Fan

To My Wife

Abstract

Automated heuristic selection and heuristic generation have increasingly attracted attention in solving combinatorial optimization problems emerging from both theory and practice. This thesis presents a heuristic generation algorithm, called Suboptimum- and Proportion-based On-the-fly Training (SPOT), which can enhance existing heuristics with the aid of instance-specific information.

By making use of the proposed "sample-learn-generate" framework, SPOT samples small-scale *subproblems*, initially. Then, it collects the *instance-specific* information from the *suboptima* of the subproblems by the means of machine learning. Lastly, it generates new heuristics by modifying existing heuristics and data structures.

In the development of SPOT, two standards were incorporated to regulate the problem input and the machine learning data. The software implementation was done in Java, with two external development libraries, the HyFlex and the Weka. In terms of testing, two well-known NP-Complete combinatorial optimization problem domains were employed: the Traveling Salesman Problem (TSP) and the permutation Flow-Shop scheduling Problem (FSP).

Each generated heuristic was tested with the TSP and the FSP domains.

To verify the result of using SPOT, one of the winners of the international hyper-heuristic competition CHeSC 2011, named PHunter, was tested with the generated heuristics by SPOT. In the TSP, adapting SPOT gave little betterment, but in FSP, the improvements were significant. It increased the overall score of the PHunter from 20.5 to 43 (out of 50). Indeed, it also outperformed the best records in CHeSC 2011: 32 by AdaptHH, 29.5 by ML and 26 by VNS-TW.

Publications

Journal Papers

Published

1. Xue, F., Chan, C. Y., Ip, W. H., & Cheung, C. F. (2011). A learning-based variable assignment weighting scheme for heuristic and exact searching in Euclidean traveling salesman problems. *NETNOMICS: Economic Research and Electronic Networking, 12*, 183 – 207. doi:10.1007/s11066-011-9064-7

2. Chan, C. Y., Xue, F., Ip, W. H., & Cheung, C. F. (2012) A Hyper-heuristic inspired by Pearl Hunting. In Y. Hamadi & M. Schoenauer (Eds.), *Learning and intelligent optimization* (pp. 349–353). Lecture Notes in Computer Science. Springer-Verlag. doi: 10.1007/978-3-642-34413-8_26

Ready for submission

1. Xue, F., Chan, C. Y., Ip, W. H., & Cheung, C. F. A hybrid variable depth search approach to personnel scheduling problems.

Conference Presentations

1. Xue, F., Chan, C. Y., Ip, W. H., & Cheung, C. F. (2010) Towards a learning-based heuristic searching reform scheme. 24th European Conference on Operational Research (EURO2010), July 11–14, Lisbon, Portugal.

2. Xue, F., Chan, C. Y., Ip, W. H., & Cheung, C. F. (2010) A learning-based variables assignment weighting scheme for heuristic and exact searching. 1st International Conference on Computational Logistics (ICCL2010), September 20–22, Shanghai, China.

3. Xue, F., Chan, C. Y., Ip, W. H., & Cheung, C. F. (2011) Pearl Hunter: a cross-domain hyper-heuristic that compiles iterated local search algorithms. The OR Society's 53rd Annual Conference(OR53), September 6–8, Nottingham, United Kingdom.

4. Xue, F., Chan, C. Y., Ip, W. H., & Cheung, C. F. (2012) A Hyper-heuristic inspired by Pearl Hunting. 6th Learning and Intelligent OptimizatioN Conference (LION6), January 16–19, Paris, France.

Acknowledgements

My most sincere thanks go to my PhD supervisors and mentors, Dr. Chan, Dr. Ip and Dr. Cheung, for supporting me in these past five years. Dr. Chan is a very sincere man and devotes himself to both theoretical and practical engineering. He gave me the freedom to pursue my interests without restriction. I am grateful to Dr. Ip, too. He introduced me to this research and inquired, discussed and enlightened me on the research very often. Dr. Cheung is supportive on both research and daily life and also provided many in-depth and insightful discussions about the research. I hope that I could be as lively and liberal as Dr. Chan and be as enthusiastic and energetic as Dr. Ip and Dr. Cheung someday. I also thank the members of my PhD committee for their helpful questions and suggestions in general.

I would also like to thank Prof. Fan and Prof. Zhu at Civil Aviation University of China. They inspired me in some related areas before the research, and have been supporting me on my daily life up-to-date. I also thank my colleagues Jackson Tang, Wang Lei, Qian Chen and Chen Qing for many interesting and insightful discussions. I will forever be thankful to my wife and my family for long-time and unconditional support, encouragement and love.

I also thank the developers, contributors and maintainers of the algorithm development libraries HyFlex and Weka, which provided stable and effective interfaces and functions on hyper-heuristic manipulation and stochastic machine learning. I also appreciate the developers, contributors and maintainers of the open source projects Notepad++ and PSPP, which offered rich and free functionality in the implementation and the data analysis of this research.

The work described in this research project was substantially supported by a grant from the Department of Industrial and Systems Engineering, The Hong Kong Polytechnic University (Grant No. RP1Z).

Table of Contents

List of Tables

List of Figures

List of Symbols

Notation	Description
$\vec{A}$	as the set of all the possible attribute vectors of solutions to a combinatorial optimization problem. 68, 69, 71–74
a	as an attribute in supervised learning. 52 as the cardinal number of a set of a limit number of paramters. 77
$\vec{a}$	as a vector of the attributes in supervised learning. 52, 55 as a vector of attributes of a solution to a combinatorial optimization problem. 68, 69, 71, 72, 74
ag	as an attribute generation for a hyper-heuristic. 68, 69, 72, 73
$\aleph_0$	as the cardinality of the natural numbers. 75–77, 87, 88

Notation	Description
$\mathbb{C}$	as the set of all countable numbers. 76, 77
C	as the number of cities in a TSP. 7, 37, 106 as the class in supervised learning. 52
c	as a city in a TSP. 7, 36, 37, 106, 108, 109, 111
cov	as the percentage of the edge candidate sets covering the edges in a known suboptimum. 111–117, 121, 127, 196
D	as a constant of "characteristic of a frontier". 15, 16 as a discrete domain of a variable in a combinatorial optimization problem. 66, 72
$\bar{d}$	as the average depth of optimal assignments in a set of predicted priority lists of assignments. 112–117, 121, 196
d	as the distance between two cities in a TSP. 7, 36, 106, 109, 111
F	as the F value of one-way ANOVA. 115, 116, 120, 143, 144, 148, 149, 197, 200, 201

Notation	Description
f	as the objective function in a combinatorial optimization problem. 66, 67
fl	as the relationship of following between two jobs in an FSP. 134, 135, 137, 138, 140
G	as a variable of the measurement scale of fractal coastline. 15, 16
Γ	as the set of possible values of confidence levels of predicting a assignment. 74, 77
γ	as the predicted value (confidence) of appearing in the optimum for an assignment. 74, 111
H	as the set of low-level heuristics for a hyper-heuristic. 68, 69, 73–77
h	as a low-level heuristic for a hyper-heuristic. 68, 69, 73, 74
hc	as a hard constraint in a combinatorial optimization problem. 67
J	as a job in an FSP. xvii, xxii, 130, 134, 137, 140
k	

Notation	Description
	as the number of maximum swaps for a tour for a TSP. 33
	as the size of k-quardrant candidate set. 103
L	as the estimate of length of seacoast. 15
l	as the learning in a hyper-heuristic. 68, 69, 72, 73
M	as a positive constant prefactor of fractal coastline. 15, 16
m	as the number of machines in an FSP. xvii, 7, 132, 135–138, 150, 155
n	as the number of cities in a TSP. xvi, 7, 10–12, 22, 24, 35, 36, 38, 40, 57, 71, 106–110, 120, 122, 123
	as the number of jobs in an FSP. 7, 130, 132, 134–137, 145, 150, 155
	as the number of variables in a combinatorial optimization problem. xvi–xix, xxii, xxiii, 10, 48, 63–67, 72–75, 77, 79, 81–83, 85, 87, 88, 94, 95, 104, 108, 110, 112–116, 119, 138, 140–144, 147, 156, 180, 187, 196, 199

Notation	Description
	as the size of a program in genetic programming. 49
	as the number of attributes. 52, 55
na	as the number of attribute vectors for a hyper-heuristic. 68, 72, 74, 77, 87, 88
nc	as the number of hard constrains in a combinatorial optimization problem. 67
nh	as the number of low-level heuristics in a hyper-heuristic. 68
O	as the big O notation, bounded above by another function asymptotically. 10, 24, 67
Θ	as the big Θ notation, bounded by another function asymptotically. 48, 49, 77
$\mathcal{P}$	as all of the NP optimization problems. 9
P	as the function of probability. 34, 36, 37, 52, 85
p	as the number of possible assignments in a group of competitor. 94–96, 108, 109, 136
	as the significance of one-way ANOVA. xvi, xvii, xx, 115, 116, 120, 121, 125, 126, 143, 144, 148, 149, 153, 197, 200, 201

Notation	Description
π	as a performance measure for hyper-heuristics. 68, 69, 73
$\mathbb{R}$	as the set of all real numbers. 67–69, 71, 73
ra	as the number of raw attributes defined for each assignment. 94–96, 109, 137
r	as the appriximation factor. 9, 10 as the function of resemblance between two solutions. xvii, 25, 79, 80, 83, 107, 108, 111, 136, 140–144, 148, 155, 156, 161, 162, 199
S	as the solution space (set of feasible solutions) to a combinatorial optimization problem. 66–69, 71–73, 79
s	as a feasible solution to a combinatorial optimization problem. 67, 69, 71, 73, 79
sam	as the sampling procedure in the SPOT hyper-heuristic. 72, 73
Σ	as a set of U/EA^2 solutions mapping from the fesiable solutions to a combinatorial optimization problem. 71–73
σ	as a U/EA^2 value of a fesiable solution to a combinatorial optimization problem. 71, 73

Notation	Description
T	as a bijection of transformation from a fesible solution to a U/EA^2 solution. 70–73
t	as the processing time of a job on a machine in an FSP. 7, 135, 137
τ	as the quantity of pheromone on an edge in ant colony approaches. 36, 38
V	as the set of all possible solutions in a combinatorial optimization problem. 66, 67
v	as an assignment (valuation) for a variable in a combinatorial optimization problem. 66
v	as the set of all assignments for a variable in a combinatorial optimization problem. 66
X	as the set of variables in a combinatorial optimization problem. 66, 70
x	as a variable in a combinatorial optimization problem. 66, 70 as a value of raw attibute. 95, 96 as the x coordinate of a city in a TSP. 109
y	as the y coordinate of a city in a TSP. 109

Notation	Description
$\mathbb{Z}$	as the set of all integers. 66, 76

List of Acronyms

Notation	Description
APX	Approximable. 9
BBO	Black Box Optimization. 13, 43
CHeSC	Cross-domain Heuristic Search Challenge. iv, xvi, xviii, 21, 45, 90, 99, 103, 104, 118, 119, 123–125, 131, 132, 146, 147, 152, 153, 167–169, 172, 198, 202
DIMACS	Center for Discrete Mathematics and Theoretical Computer Science. xxi, 14, 17, 33, 40
DSAFO	Dynamic Scheduling Agents with Federation Organization. 39
EDAs	Estimation of Distribution Algorithms. 32–36, 39, 54, 61, 63, 166

Notation	Description
EHBSA	Edge Histogram Based Sampling Algorithm. 35
FSP	permutation Flow-Shop scheduling Problem. iii, iv, xvi–xix, xxii–xxiv, 7, 9, 25, 26, 88, 129–139, 141, 145, 146, 151–155, 160–163, 167, 168, 171, 172, 178, 187–192
GCC	GNU Compiler Collection. 98
GRASP	Greedy Random Adaptive Search Procedure. 32, 34, 39, 40, 57
HK	Held-Karp. 12, 14, 33, 102
HyFlex	Hyper-heuristics Flexible framework. iii, viii, xv, xvi, xviii, xxii, xxiii, 89–92, 97–99, 101, 103–105, 109, 112, 114, 119, 122, 123, 125–127, 129, 131, 132, 135, 145, 147, 150, 152–154, 167, 169, 170, 172, 196
LKH	Lin-Kernighan-Helsgaun. 33

Notation	Description
LLH	low-level heuristic. xv–xvii, xxiii, 41, 42, 44–46, 48, 49, 57, 61, 62, 65, 68, 69, 73–79, 83–86, 90–93, 98, 99, 104, 105, 108, 110–112, 117–125, 127, 128, 131–133, 138, 144–157, 165, 169, 173, 200
NFL	No-Free-Lunch. 13, 15, 23, 43, 164, 173
NP	Non-deterministic Polynomial time. 7–9
NP-Complete	Non-deterministic Polynomial time-Complete. iii, 7–10, 12, 21, 33, 35, 38, 41, 67, 68, 87, 88, 99, 102, 130, 159, 160, 164, 171, 172
NP-hard	Non-deterministic Polynomial time-hard. 8, 9, 12, 23, 64, 102, 130, 164
P	Polynomial-time. 7–9
PHunter	Pearl Hunter. iv, xvi, xviii, xxii, xxiii, 21, 42, 45, 97, 99, 111, 118, 119, 123–126, 138, 139, 146, 147, 152–154, 157, 167, 172, 197, 198, 201, 202
SAT	Boolean Satisfiability Problem. 44, 45, 49

Notation	Description
SPOT	Suboptimum- and Proportion-based On-the-fly Training. iii, iv, xvi–xviii, xxii, xxiii, 15, 24–26, 29, 59–62, 65, 66, 70–74, 77–81, 83, 85–87, 89, 92–94, 96–99, 101, 105–111, 113, 114, 117–119, 121–129, 133, 136–142, 144, 146–148, 151–157, 159–167, 169–173, 186, 193, 194, 196–199, 201, 202
TSP	Traveling Salesman Problem. iii, iv, xv, xvi, xviii, xix, xxi–xxiii, 7, 9–12, 14, 17, 22–26, 33, 35, 36, 38, 40, 47, 49, 57, 60, 71, 80, 83, 84, 86, 87, 101–110, 113, 117, 118, 121–126, 128, 130, 131, 140, 145, 160–163, 165–169, 172, 175–177, 180, 181, 183–186
U/EA^2	Unconstrained and with Equinumerous Assignments and Equinumerous Attributes. 66, 71, 73, 78, 79, 81, 106, 107, 110, 133, 136, 137, 161
U/EA	Unconstrained and with Equinumerous Assignments. xxii, 61, 63, 66, 70–73, 78–80, 83, 87, 92, 106, 107, 133, 135, 155, 160–162, 164, 169

Notation	Description
VLSI	Very-Large-Scale Integration. 102
Weka	Waikato Environment for Knowledge Analysis. iii, viii, xv, 51–54, 82, 89–93, 95–97

Chapter 1

Introduction to the SPOT Algorithm

> Many [geographical curves] are statistically "self-similar," meaning that each portion can be considered a reduced-scale image of the whole.
>
> *How Long is the Coast of Britain?*
>
> Benoît B. Mandelbrot

Many — if not all — human activities involve searching for ways to achieve particular goals. Individuals and organizations are called rational if they maximize their benefits or minimize inconvenience in equivalent measures. As a result, optimal decisions are highly demanded in many practical as well as theoretical problems.

1.1 Optimizations from Ancient to Modern Times

Since antiquity, many problems on decision making emerged from human social life, and were in the form maximizing benefits. Those successful solvers became well-known representatives of human intelligence and rationality. The Roman poet Virgil (29-19B.C./1904, lines 1.365–1.368) recorded a legendary story in the foundation of Carthage by Queen Elissa (Dido of Carthage) around 800 B.C.:

> Dēvēnēre Locōs, ubi nunc ingentia cernis
> moenia surgentemque novae Karthāginis arcem,
> mercātīue solum, factī dē nōmine Byrsam,
> taurīnō quantum possent circumdare tergō.
>
> [English (Virgil, 29-19B.C./1893, p. 14):
> So to the place they came, where now thou spyest
> The lofty walls and rising citadel
> of new-built Carthage, and of land they bought —
> Called Byrsa from their bargaining — so much
> As with a bull's hide they might compass round.]

This *isoperimetric problem* of enclosing the maximum area with a given perimeter is now also known as Dido's problem (e.g., Merrill, 1919). The solution for this case, with the Mediterranean coast as a given edge, is a semicircle (Hackley, 1847, Appendix I, Theorem IV). Someone said that the cunning Queen Elissa cut the bull hide into very narrow strips and circumscribed a

maximized size of land in a semicircle[1]. Some others, such as R. B. Smith (1913, p. 14), argued that the Greek legend was based on the superficial resemblance of the Phoenician word "Bozra" (a fortress) to the Greek "Byrsa" (an ox-hide). In addition, archaeological evidence shows that the city wall of Carthage[2] was not of circular arc shape. Although the authenticity of the ox-hide legend is debatable, it is still clear that the idea of Dido's maximization appeared for more than two millennia.

Another historical story from the Kingdom *Ch'i* in the Warring States Period, about 350 B.C., of ancient China was recorded by Sima (91B.C./2010, pp. 01.761–01.762):

> 忌數與齊諸公子馳逐重射。孫子見其馬足不甚相遠，馬有上、中、下輩。於是孫子謂田忌曰：「君弟重射，臣能令君勝。」田忌信然之，與王及諸公子逐射千金。及臨質，孫子曰：「今以君之下駟與彼上駟，取君上駟與彼中駟，取君中駟與彼下駟。」既馳三輩畢，而田忌一不勝而再勝，卒得王千金。

> [English (Ssu-ma, 91B.C./1994, pp. 39–40): [*Ch'i*'s general] *T'ien Chi* 田忌 raced horses and gambled heavily with the Noble Scions of *Ch'i* several times. [His guest] *Sun Tzu* [*Sun Pin* 孫臏] noticed that the horses' speed was not much different and that the horses fell into high, middle and low grades. After this, *Sun Tzu* told *T'ien Chi*, "Just bet heavily, My Lord, and I can make you the winner."

[1] See Elissar, Dido, the Queen of Carthage and her city, http://www.phoenicia.org/elissardidobio.html.

[2] See History Channel's TV program *Engineering an Empire: The Carthage*.

> *T'ien Chi* confidently agreed and bet a thousand *chin*[3] with King [*Wei* 威, r. *378–343 B.C.] and the Noble Scions [of *Ch'i*] on a race. Just before the wager *Sun Tzu* said, "Now match their high-grade horses with your low-grade horses, take your high-grade horses to match their middle-grade horses and take your middle-grade horses to match their low-grade horses."
>
> After they raced the three grades [of horses], *T'ien* lost once but won twice and eventually gained the king's thousand *chin*.]

The military strategist *Sun Pin* (or *Bin*), author of *Sun Bin's Art of War*, suggested to General *T'ien Ch'i* a discrete placement strategy for horse-racing, as shown in Figure 1.1. *Sun* took advantage of the best combination[4] of matching and made *T'ien* the winner. After that, King *Wei* questioned *Sun* on the arts of war and made him his counselor.

Minimization or maximization problems solved with modern mathematics can be traced back to the 17th century. One well known problem was Bernoulli (1696)'s *brachistochrone* (Woodhouse, 1810/1964, p. 3), i.e. fastest descent curve without friction, as shown in Figure 1.2:

> Datis in plano verticali duobus punctis A et B, assignare mobili M viam AMB, per quam gravitate sua descendens, et moveri incipiens a puncto A, brevissimo tempore perveniat ad alterum punctum B.
>
> [English (D. E. Smith, 1929, p. 645): If two points A and B are

[3] Note: Weight unit, 1 *chin* = 256.26 g.
[4] It should be noted that this matching plan is not an equilibrium.

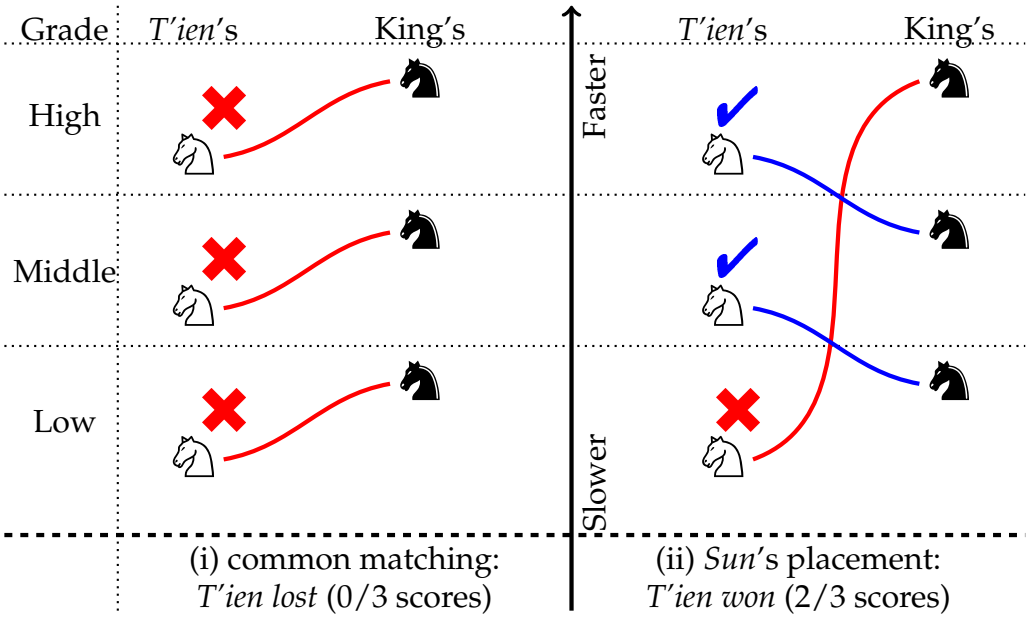


Figure 1.1: *Sun*'s placement for *T'ien Chi*'s horses racing

given in a vertical plane, to assign to a mobile particle M the path AMB along which, descending under its own weight, it passes from the point A to the point B in the briefest time.]

The name brachistochrone was derived from Greek $\beta\rho\acute{\alpha}\chi\iota\sigma\tau\varsigma$ (*brachistos*, the shortest) and $\chi\rho\acute{o}\nu\varsigma$ (*chronos*, time or delay). Six mathematicians, Newton, Leibniz, Von Tschirnhaus, L'Hôpital, his elder brother Jacob Bernoulli and Bernoulli himself (Gerdts, 2012, p. 1; Stillwell, 2010, p. 259), (independently) found a solution — the cycloid[5]. Stillwell (2010, p. 259) regarded Jacob Bernoulli's solution as the most profound and the first major step in the development of the calculus of variations.

During the last decades, the widespread use of computers enabled a lot of optimization methods to be used that otherwise would be too tedious and

[5]Galileo proved a circular arc is better than strait line for brachistochrone in 1638, but failed to recognize a cycloid (Babb & Currie, 2008).

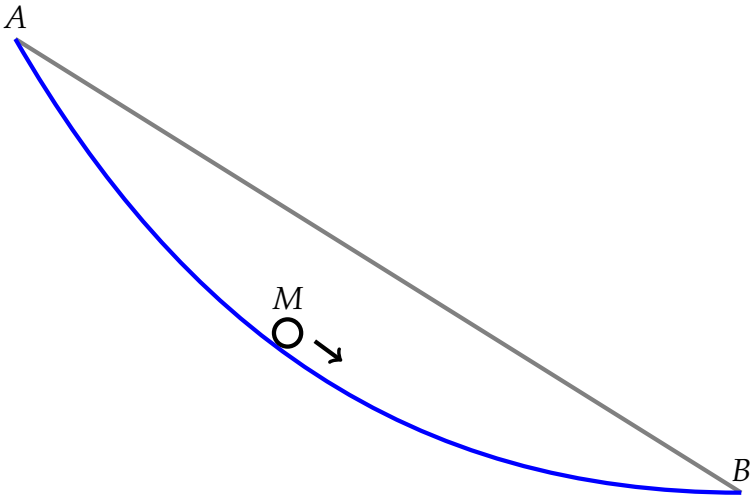


Figure 1.2: The brachistochrone problem

time-consuming for human planners. With a steadily developing computer technology nowadays, the increasing machine power becomes an indispensable tool (Kleunen, 1976) and makes the optimization process more and more efficient. In the contemporary world, almost all branches of engineering have been increasingly dependent on computerized techniques (Kassim & Cadbury, 1996).

1.2 Combinatorial Optimization: Problems and Algorithms

This thesis focuses on a class of problems encoded in discrete values called *combinatorial optimization problems*. In a combinatorial optimization problem, the aim is to look for an object from a finite (or possibly countably infinite) set (Papadimitriou & Steiglitz, 1982, p. 2). A typical object can be an integer number, a subset, a permutation or a graph structure (C. Blum & Roli, 2003). The best object is called the *optimal solution* or *optimum*, and the objects very close to the optimal solution are called *suboptimal solutions* or *suboptima*.

Examples of combinatorial optimization problems are the *Traveling Salesman Problem* (TSP) and the *permutation Flow-Shop scheduling Problem* (FSP). The definitions of the TSP and the FSP are quoted as follows, respectively.

> In the traveling salesman problem, a set $C = \{c_1, c_2, \ldots, c_n\}$ of cities is given, and for each pair (c_i, c_j) of distinct cities, a distance $d(c_i, c_j)$. The target is to find a Hamiltonian cycle (also known as a Hamiltonian circuit) of all the cities that minimizes the tour length (D. S. Johnson & McGeoch, 1997).

> Given n jobs to be processed on m machines in the same order, the process time of job i on machine j being $t_{i,j}$ ($i = 1, 2, \ldots, n$; $j = 1, 2, \ldots, m$), find the sequence of jobs such that the total elapsed time (makespan) is minimized (Nawaz, Enscore, & Ham, 1983).

The TSP and the FSP seem easy to describe and validate, but they have been proved to be in the NP-Complete (Non-deterministic Polynomial time-Complete) class — a very difficult class of problems (Papadimitriou, 1977; Garey, D. S. Johnson, & Sethi, 1976). In computational complexity theory, NP[6] (Non-deterministic Polynomial time) is the set of decision problems solvable in polynomial time on a non-deterministic Turing machine. P (Polynomial-time) is the complexity class containing decision problems which can be solved by a deterministic Turing machine using a polynomial amount of computation time. Equivalently, solutions to NP problems can be verified

[6] See http://qwiki.stanford.edu/index.php/Complexity_Zoo:N.

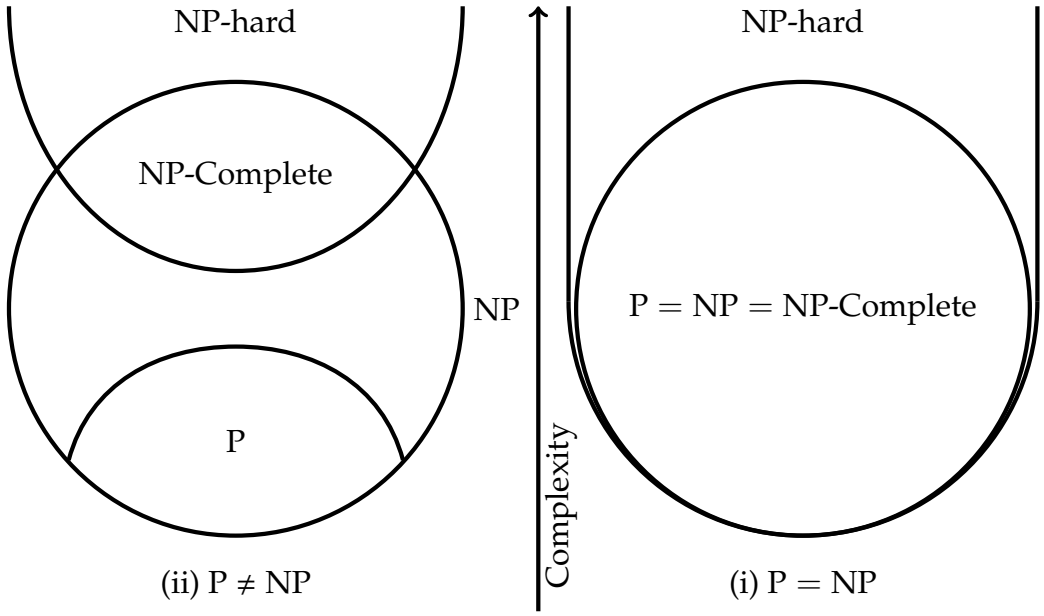


Figure 1.3: Euler diagram for classes of P, NP, NP-Complete and NP-hard

by a deterministic Turing machine in polynomial time. A problem is NP-Complete, if and only if every other NP problem is *reducible* to this problem (Garey & D. S. Johnson, 1979, p. 13). Another common term, NP-hard (Non-deterministic Polynomial time-hard), represents the class of problems that are at least as hard as problems in NP-Complete, as shown in Figure 1.3 (Wikipedia, 2012b). If any one NP-Complete problem can be generally solved in polynomial-time on a deterministic Turing machine, so can every NP problem.

The relationship "P versus NP?" is still unsolved, and the Clay Mathematics Institute is offering[7] a US$ one million prize for the first correct proof or disproof. However, most researchers believe P ≠ NP. For example, in Aaronson (2005)'s poll, 61 out of 100 theorists thought P ≠ NP, 9 thought P = NP,

[7] As one of the seven Clay Mathematics Institute's Millennium Prize Problems, "P versus NP?" problem was announced on May 24, 2000 at Collège de France and is still open in 2012. See http://www.claymath.org/millennium/.

22 offered no opinion and 8 offered other opinions. In this thesis, P ≠ NP is assumed. As a result, no polynomial-time (A. L. Blum & Rivest, 1992) or sub-exponential time (Woeginger, 2003) algorithm can exactly solve any NP-Complete problem, in the worst case. In other words, for an NP-Complete problem, the solution process will typically face either

- an *explosive* running time for an optimal solution, or
- *suboptimal* solutions for a reasonable running time.

In fact, to guarantee finding a suboptimum for an NP-Complete problem is also very difficult. Ausiello et al. (1999, p. 93) defined APX (Approximable) as the class of all NP optimization problems $\mathcal{P}$ such that, for some factor $r \geq 1$, there exists a polynomial-time r-approximation algorithm for $\mathcal{P}$. Ausiello et al. (1999, p. 94) proved that a TSP holding *triangular inequality* does not belong to APX unless P = NP, and Papadimitriou and Vempala (2006) proved such a TSP is NP-hard to approximate, if the ratio is less than 220/219. Williamson et al. (1997) proved that an FSP is NP-hard to approximate with a factor less than 5/4.

Exact algorithms for solving combinatorial optimization are those which guarantee finding an optimal solution in a finite amount of time. Examples of exact algorithms include:

branch-and-bound which systematically enumerates all feasible solutions in a tree and drops subtrees on-the-fly by lower and upper bounds (Lawler & Wood, 1966),

Lagrangian relaxation based methods such as Held and Karp (1970)'s 1-

tree for TSP,

integer (linear) programming based methods [8] such as the cutting-plane algorithm (Nemhauser & Wolsey, 1988, pp. 367–378) and the branch-and-price (Barnhart, Johnson, Nemhauser, Savelsbergh, & Vance, 1998), based on branch-and-bound and column generation,

exponential neighborhood search with an exponential setup time followed by a polynomial (e.g., in $O(n^3)$) time local search (Ahuja, Ergun, Orlin, & Punnen, 2002; Gutin & Glover, 2005),

recursive exhaustive search which enumerates all possible combinations and is not the best option usually.

The exact approaches mainly encounter the first difficulty. However, when the number of variables is limited, most exact algorithms — sometimes even exhaustive search algorithms — can be very effective and efficient for problem solving, see Applegate, Bixby, Chvátal, and Cook (2001)'s branch-and-cut for small-scale TSPs ($n < 10^3$).

Heuristic (also known as *approximate*[9] and *incomplete*) optimization algorithms only concentrate on a (small) subset of the solution space, usually by experience-based techniques, and cannot guarantee the optimality of the results. However, many heuristic optimization algorithms can find "good enough" solutions in a "small enough" computation time. Examples of heuristic optimization algorithms are:

[8] Modeling or transforming a combinatorial optimization problem into integer programming does not help on the complexity. The integer programming itself is NP-Complete in the strong sense (Karp, 1972), even each integer belonging to the set {0,1} (Sahni, 1974).

[9] It should be noted that an approximate algorithm is called r-approximation if and only if it guarantees its results are no more than r times the optimum.

first-come, first-served (or first-in-first-out): always selects the first available component or assignment for the eventual result,

greedy construction: a step-by-step incremental construction, with a greedy rule in each step, such as the *multiple-fragment merge* for TSPs (Bentley, 1992),

alpha-beta pruning: an approximate version of branch-and-bound with inexact branch reduction, see (Knuth & R. W. Moore, 1975),

meta-heuristics: high-level strategies for guiding a search (Glover, 1986) according to feedback from the objective function, previous decisions and prior performance (Stützle, 1998). More details can be found in Section 2.1,

random: a return of a randomly generated solution, not the best option usually.

The computations of heuristic optimization algorithms are usually in polynomial-time. However, there might possibly exist some exponential heuristic optimization algorithms, such as an impractical local search for the best $0.5n$-edge-swap solution in a TSP.

Integrations of exact and heuristic optimization algorithms (Puchinger & Raidl, 2005) can be either exact or heuristic. An example of exact integrations was given by French, Robinson, and Wilson (2001)'s branch-and-bound for *maximum satisfiability*. They employed a steady-state meta-heuristic (evolutionary algorithm) to repeatedly suggest the fittest solutions from hanging nodes in order to guide branching. The new integration still guaranteed to

find an optimal solution and was reported to be better than a single branch-and-bound or an evolutionary algorithm. A heuristic example by Cotta and Troya (2003) also integrated the branch-and-bound and evolutionary algorithm, though in a different way. They embedded branch-and-bound as a low-level operator in the evolutionary algorithm. Their approach was heuristic, because its main framework was a heuristic and did not guarantee an optimal result.

In large-scale NP-Complete optimization, exact and heuristic optimization algorithms are two indispensable forces for solving problems. The optimum of a problem can be found by the *squeeze theorem*, if the *greatest lower bound* by exact algorithms and the *least upper bound* by heuristic optimization algorithms become equal. For example, Applegate, Bixby, Chvátal, Cook, et al. (2009) improved the lower bound to a known feasible solution found by heuristics and thus certificated an optimum for a large-scale TSP instance ($n = 85,900$) in 286.2 CPU-days for a 2.4 GHz CPU.

One algorithm is said to be *dominated* by another if it generally returns a worse (or the same) solution while costs more computation time, and vice versa for *non-dominated*. Only *a few* non-dominated exact algorithms exist for a problem, at the most, because every exact algorithm guarantees the same optimum. On the other hand, there are *a lot of* non-dominated heuristic algorithms for each NP-hard combinatorial optimization problem. For example, Figure 1.4 shows some non-dominated heuristic optimization algorithms for the 10000-city random Euclidean TSP, where the Held-Karp (HK) bound is a lower bound. In practice, the economic importance of many combinatorial optimization problems stimulates increasing demands for

more non-dominated algorithms. This is also a cause leading to the thriving of heuristics studies.

Although many heuristics aim at being cross-domain, besides being non-dominated, there is a deadly pitfall from the No-Free-Lunch (NFL) theorems (Wolpert & Macready, 1997). There are two NFL theorems relating to *the set of all possible problems* and *the set of all possible algorithms*, respectively:

Theorem 1 (No-Free-Lunch Theorem 1)**.** *Over all possible problems, no search algorithm is better than any other algorithm, on average.*

Theorem 2 (No-Free-Lunch Theorem 2)**.** *Over all possible search algorithms, no search problem is harder than any other problem, on average.*

The NFL theorems have been proved in different ways, such as in Wolpert and Macready (1997) and Culberson (1998). Though the theorems can be criticized as too ideal, such as the condition of "all possible problems". The NFL theorems mean that no algorithm can perform statistically better than a random search of possible solutions on average, if no problem-specific information is considered (also known as the Black Box Optimization, or BBO).

In fact, handling the problem-specific characteristics is important for a general-purpose optimization algorithm to stay away from the pitfall of NFLs (Smith-Miles, 2008a). There are at least two options for an optimization algorithm, theoretically, to keep a safe distance from the pitfall of the NFLs:

- To become a well-designed problem domain-specific (*ad-hoc*) algorithm (Burke, Kendall, et al., 2003, pp. 459–460), and

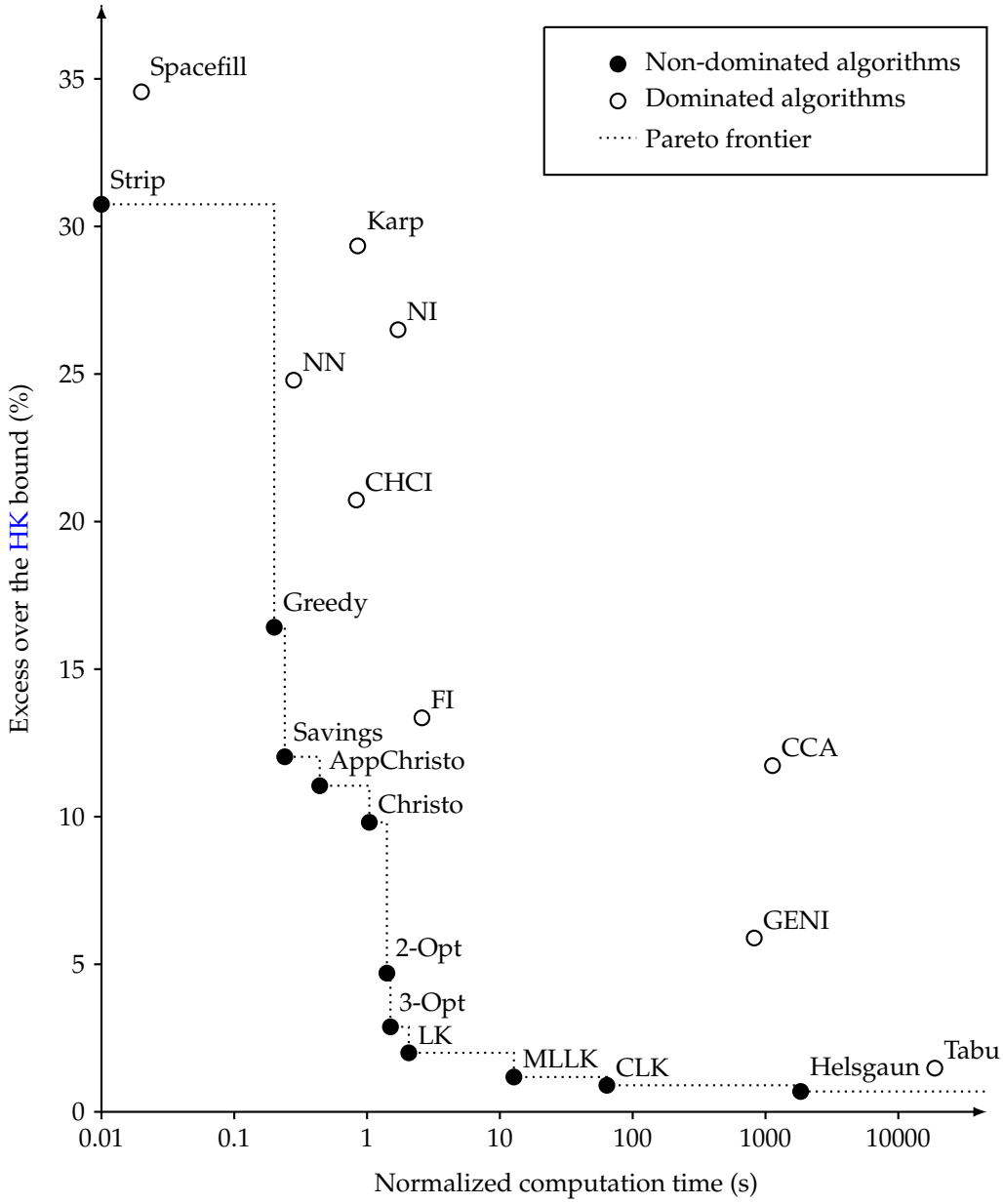


Figure 1.4: Examples of non-dominated and dominated heuristics for solving the 10000-city Random Uniform Euclidean Instances in the DIMACS TSP Challenge (D. S. Johnson & McGeoch, 2002, p. 376)

- To discover and to make use of domain-specific, problem data set-specific and/or instance-specific characteristics dynamically, from case to case,

where most of the successful cross-domain heuristics as well as exact methods belong to the latter class (e.g., Poli & Graff, 2009).

In this study, the SPOT methodology is proposed for *cross-domain* combinatorial optimization problems. The SPOT methodology obtains *instance-specific* information by machine learning to generate *new* heuristics or to modify existing heuristics towards a *non-dominated* direction. Meanwhile, the instance-specific information keeps the SPOT method safe from the pitfall of the NFL theorems.

1.3 Research Motives and Background

Some extremely difficult real-world problems can be successfully approximated to the best results very efficiently by investigating *instance-specific* information in a *portion* of the whole problem. One example is the *self-similarity* in fractals (Hutchinson, 1981). A self-similar object is exactly or approximately similar to a part of itself. Mandelbrot (1967) presented such a well-known approach to measure the length of the coastline of Great Britain, see Figure 1.5 (Wikipedia, 2012a). Mandelbrot cited Richardson (1961)'s estimate of length

$$L(G) = MG^{1-D}, \tag{1.1}$$

Table 1.1: Examples of the characteristic D on the map

D	Geographic boundary
1.00	An extreme value, a frontier that looks straight on the map
1.25	The west coast of Great Britain, which looks like one of the most irregular seacoasts in the world
1.15	The land frontier of Germany in about 1899 A.D.
1.14	The land frontier between Spain and Portugal
1.13	The Australian coast
1.02	The coast of South Africa, a coast selected as looking one of the smoothest in the atlas.

where M is a positive constant prefactor, G is a variable of the measurement scale and $D \geq 1$ is an instance-specific constant of the fractal dimension or a "characteristic of a frontier, may be expected to have some positive correlation with one's immediate visual perception of the irregularity of the frontier". Hutchinson (1981) listed the characteristics of some geographic boundaries, as shown in Table 1.1. Geographic curves in the real world are so involved in their detail that their lengths are often infinite (or indefinable). Now, however, the whole coastline can be estimated by measuring the "characteristic" D through investigation of a small portion of the whole seacoast.

Another example is the *sampling*. In statistics and survey methodology, sampling consists of selecting some part of a population to observe so that one may estimate something about the whole population (Thompson, 2002, p. 1). For example, the mean and the standard deviation of a population of real numbers following an unknown normal distribution can be unbiased estimatedly through the mean and the standard deviation of a random sample, respectively. A sample is *representative* only when the sampling process is both *accurate* (with a mean of errors no more than a given threshold) and *reproducible* (with a variance of errors no more than another given threshold)

$G = 200\text{km}$
$L(G) \approx 2,400\text{km}$

$G = 100\text{km}$
$L(G) \approx 2,800\text{km}$

$G = 50\text{km}$
$L(G) \approx 3,400\text{km}$

Figure 1.5: Britain fractal coastline

(Petersen, Minkkinen, & Esbensen, 2005). The success of the estimations lies in the representativeness, which relies on the *characteristics* of the population. Some well-known applications of sampling include census and production testing (e.g., Milor & Sangiovanni-Vincentelli, 1994).

What attracts interest in this thesis are the instance-specific characteristics (usually) hidden in combinatorial optimization problems and the methods of finding them from a proportion of a large-scale problem. Take two benchmark instances of TSP as an illustration: P_1 (the `E100.0` from the DIMACS TSP Challange[10]) with randomly generated cities and P_2 (the `xqf131` from

[10]The eighth DIMACS (Center for Discrete Mathematics and Theoretical Computer Science) Implementation Challenge (2001), see http://dimacs.rutgers.edu/Challenges/ and http://www2.research.att.com/~dsj/chtsp/.

the VLSI data set[11]) from the print circuit industry with visually regular cities, as shown in Figure 1.6. Since the visual difference in Figure 1.6 is clear, one can hope that the instance-specific characteristics can probably unveil more information than common knowledge, such as the triangular inequality. Another question is how to find the characteristics from a proportion, if possible.

In the literature, various approaches have employed instance-specific information as "characteristics" to adjust both exact and heuristic optimization algorithms for better performances for decades. There are three typical ways of adjustment, including:

algorithm selection: returns a preferred subset of ready-to-use heuristics,

parameter optimization: (also known as *parameter tuning* and *algorithm configuration*) returns numerical parameters for heuristics,

algorithm generation: returns interesting[12] (most likely unknown previously, potentially useful) heuristics.

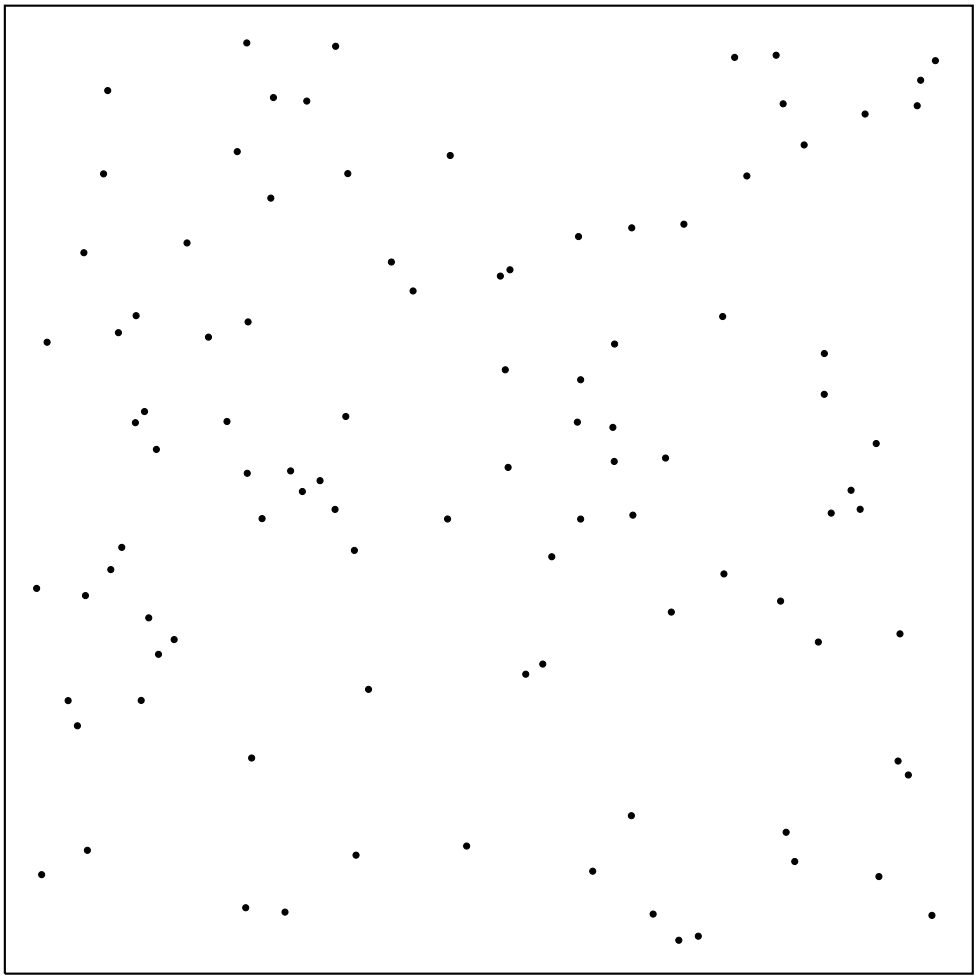
According to the nature of the object algorithm, the three typical ways can be further divided, as shown in Figure 1.7. Figure 1.7 also shows the relationships between the adjustments and two popular concepts of heuristics, *meta-heuristics* and *hyper-heuristics*[13].

Typical algorithm selection for exact algorithms are: Lobjois and Lemaître (1998)'s branch-and-bound approach which determines branching from a

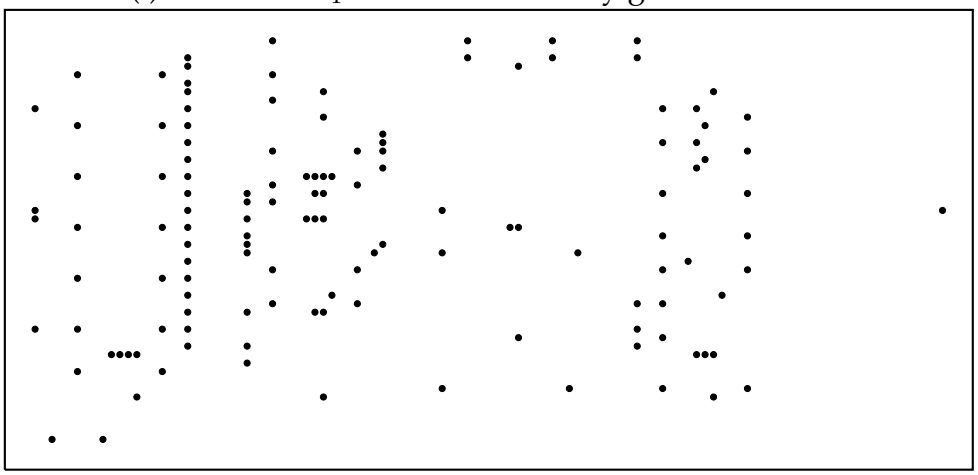
[11] See http://www.tsp.gatech.edu/vlsi/.

[12] Nevertheless, some interesting results could be dominated by existing research, such as Kotsiantis, Zaharakis, and Pintelas (2006).

[13] See Section 2.2 and Section 3.2.5.



(i) A 2D TSP P_1 with 100 randomly generated cities



(ii) A 2D TSP P_2 with 131 cities from print circuit industry

Figure 1.6: TSP Examples from different data sets

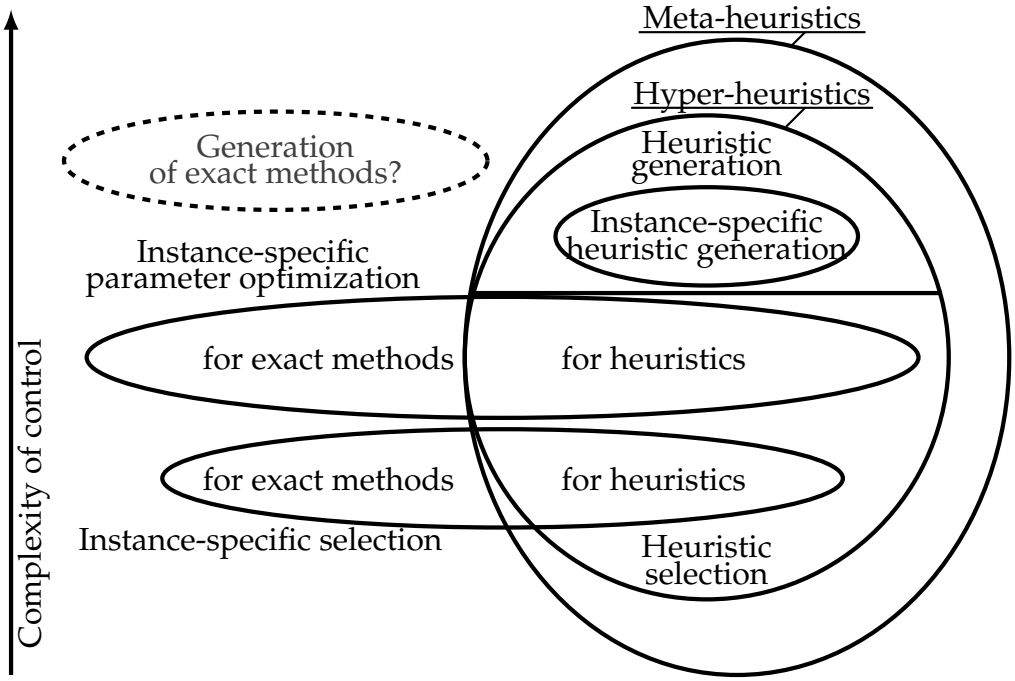


Figure 1.7: Euler diagram of instance-specific adjustments of algorithms, hyper-heuristics and meta-heuristics

number of (heuristic) options by *performance prediction* for each particular problem instance, and Leyton-Brown, Nudelman, Andrew, McFadden, and Shoham (2003)'s algorithm portfolios. Examples of parameter optimization for exact algorithms are Kadioglu, Malitsky, Sellmann, and Tierney (2010)'s *instance-specific algorithm configuration* for CPLEX, the best known *mixed integer programming* solver and Hutter, Hoos, Leyton-Brown, and Stützle (2009)'s local search-based method named ParamILS. As far as is known, no approach has been directly developed for exact algorithm generation. However, there may be some possible indirect ways, such as Ibaraki and Muroga (1970)'s *linear programming* for discovery of classifiers, and later works such as Brown, Pittard, and Park (1996)'s *optimum* split on tree nodes. Such an indirect approach can be called an "exact generation" if:

- there exists a one-to-one transformation between instances in an optimization problem domain to a set of classification problems; and
- the results of linear programming can guarantee no error (optimum) for *any* instance of classification.

Furthermore, it must cost at least exponential time (Woeginger, 2003) in general for any exact generation, if it exists, in solving an NP-Complete optimization problem. In fact, if there is a non-zero error between the training labels (from the optimum) and the predicted values of the labels (usually represented as a solution), the generation can be considered as heuristic generation in practice.

Heuristic selection, including instance-specific algorithm selection and parameter optimization for heuristics, has received increasing attention in recent years. For example, there were twenty heuristic selection algorithms submitted to the hyper-heuristic competition CHeSC (Cross-domain Heuristic Search Challenge) 2011[14] from all over the world. Each competitor had a particular strategy of making use of instance-specific and domain-specific information. Some cross-domain heuristic selection methods proved their non-dominance in certain domains, such as Chan, Xue, Ip, and Cheung (2012)'s Pearl Hunter (PHunter) hyper-heuristic which found new best-known solutions (upper bounds) for six benchmark instances in the Employee Scheduling Benchmark Data Sets[15]. The space of results of a heuristic selection is far smaller than that of heuristic generation, as discussed in Section 3.2.5.

[14]The first Cross-domain Heuristic Search Challenge, see http://www.asap.cs.nott.ac.uk/external/chesc2011/.

[15]See http://www.cs.nott.ac.uk/~tec/NRP/.

In the literature, many heuristic generation methods are based on stochastic learning techniques. Conventional stochastic learning methods belong to supervised learning, which builds a concise model of the distribution of class labels in terms of decision attributes (or predictor features) (Kotsiantis et al., 2006). Typical examples include genetic programming and decision tree (or rules) learning. Genetic programming constructs a heuristic in an online evolutionary way (Koza, 1992, pp. 79–120). The latter method typically builds a heuristic in a hierarchical structure of rules that recursively classify the class labels by construction and pruning (Murthy, 1998). Genetic programming has been the most common (Burke, Hyde, Kendall, Ochoa, Ozcan, & Qu, 2010) heuristic generation methodology.

However, many heuristic generation approaches could not generate non-dominated new heuristics in comparison to existing ones. For example, as "the best over all genetic programming work done on TSP", Keller and Poli (2007)'s genetic programming hyper-heuristic was limited to very small-scale ($n \leq 10^2$) TSP instances. In comparison, a meta-heuristic (e.g., Helsgaun, 2000) could *almost* guarantee to find an optimal solution for $n \leq 10^3$ in a few seconds. There are two main problems that make the feasibilities of many heuristic generation methods very low:

Computation time: The time cost is, in fact, always crucial for non-dominance and practicability of any algorithm. (See Figure 1.4)

Generalization: The high cross-domain capability was "the biggest challenge" (Burke, Hyde, Kendall, Ochoa, Ozcan, & Qu, 2010).

The main reason for the first difficulty is probably related to the explosive

search space of heuristics to generate, especially in large-scale NP-hard combinatorial optimization problems, see Section 3.2.5. Other reasons for the first difficulty depend on the learning methods, such as the (probably possible) non-convergence of the canonical evolution (Rudolph, 1994) for a genetic programming and the very expensive time for preparing a number of training examples for a decision tree or rule learning. Reasons for the latter difficulty also depend on the learning, such as the encoding in some problem domains (Burke, Hyde, Kendall, Ochoa, Ozcan, & Qu, 2010) for a genetic programming and the definitions of attributes and mutually exclusive classes (Bose & Mahapatra, 2001) for a decision tree or rule learning. It should be noted that the NFL theorems would be included in the second difficulty, when all the domains were claimed by an algorithm.

Xue, Chan, Ip, and Cheung (2011) presented some ideas for handling the first difficulty. They proposed a novel sampling method to resolve the first problem. The supervised class association rules learning was employed to discover the probability of assignments of being "promising" for each decision making in choosing two edges starting from a known city. It was found that the rules learned from a sampled subproblem were generally consistent with those from the master problem in the Euclidean TSP. The approach consists of four main steps:

- sampling a *proportion* (subset of variables with corresponding constraints) with a *bounded size* as a subproblem,
- solving the subproblem,
- finding new stochastic heuristic rules for the portion, and

- applying new heuristic rules back to the given problem.

The sampling stood for selecting a subset of variables from all the given variables, as well as the corresponding constraints. The bounded size on the proportion dramatically reduced the time complexity (to $O(n \log n)$). The experimental results of the effectiveness of the approximate method in Euclidean TSPs ($3.16 \times 10^3 \lesssim n \leq 10^6$) seemed very satisfactory. The time cost of the method was less than a typical iteration of the resulting heuristic when $n \geq 3.16 \times 10^4$. However, the second problem was not mentioned in Xue, Chan, et al. (2011). The decision attributes were experience-based, specific for the Euclidean TSP, and fixed. The limited attributes and finite values for each attribute made the method a heuristic selection, though stochastic learning was involved. See Section 3.2.5 for the definitions.

1.4 Objectives and Achievements of Thesis

This study aims at presenting the SPOT method, which is expected to be a practicable, instance-specific and cross-domain heuristic generation methodology. Therefore, the difficulties on time cost and generalization addressed in Section 1.3 would inevitably be encountered. The objectives are given in Section 1.4.1 in order to try to tackle the two difficulties. Section 1.4.2 presents major achievements of the thesis.

1.4.1 Objectives

The primary objective of this research is

- To develop an efficient and instance-specific methodology that carries out on-the-fly supervised learning.

There are four particular supporting objectives to fulfill the primary objective, including:

- To standardize the input combinatorial optimization problem in the methodology,
- To regulate a systematic way of compiling decision attributes for various measures in different domains,
- To define an indicator to guide the development procedure and to predict the effectiveness approximately, and
- To explore effective ways of making use of learned instance-specific information to generate new heuristics.

1.4.2 Achievements and contributions

All the objectives listed in Section 1.4.1 were achieved in this thesis. Chapter 3 presents the instance-specific *heuristic generation* approach SPOT (Suboptimum- and Proportion-based On-the-fly Training) as an extension of Xue, Chan, et al. (2011). The standard of the input problem and an indicator r of "resemblance" are given in Section 3.2.3 and examined in the two domains of the TSP and the FSP. A set of equations for generating new decision attributes based on domain-specific measures are designed to normalize the data for learning, as shown in Section 4.3. Applications and experiments on the TSP and the FSP domains are represented in Chapter 5 and Chapter 6.

Some plans for modifying existing heuristics, including mutations and local search, are also illustrated in Chapter 5 and Chapter 6.

Besides the SPOT methodology, another important theoretical contribution is the formulation of hyper-heuristics in Section 3.2.4 and the formal definitions of heuristic selection and heuristic generation in Section 3.2.5. In this thesis, the boundary between the two subclasses of hyper-heuristics is defined on the basis of the countability of the maximum number of possibly generated heuristics. The formulations, especially the formal distinction between heuristic selection and heuristic generation, are proposed for the first time, as far as is known.

1.5 Outline of Thesis

Besides this chapter, there are six more chapters in this thesis. As a continuation of the research background, state-of-the-art of hyper-heuristics, related meta-heuristics and other approaches are reviewed in Chapter 2. Among all the approaches, heuristic generation methods are the focus. Chapter 3 proposes the concept of the SPOT method and the supporting standards and indicators for it. Formal definitions of the combinatorial optimization problem, hyper-heuristics, the two subclasses of hyper-heuristics and the SPOT algorithm are also introduced in Chapter 3. An implementation is introduced on the basis of two well-known algorithm development frameworks in Chapter 4. Two test domains, the TSP and the FSP, are involved in Chapter 5 and Chapter 6, respectively. The whole methodology of developing the SPOT hyper-heuristic to the two typical combinatorial optimization prob-

lems can be found in Chapter 5 and Chapter 6. Discussion and concluding remarks are given in Chapter 7.

Chapter 2

Literature Review

Zij die wensen dat de toekomst anders is dan het verleden, moeten het verleden bestuderen.
If you want the present to be different from the past, study the past.

Ethica (*Ethics*)

Baruch Spinoza

Important approaches related to the SPOT algorithm, such as heuristic generation, sampling and machine learning, are recalled in this chapter. The families of typical meta-heuristics are reviewed in Section 2.1. Some meta-heuristic methods involving sampling techniques are introduced in detail. Section 2.2 surveys hyper-heuristic approaches, including heuristic selection and heuristic generation methods. The supervised machine learning, or classification, and related techniques in machine learning are reviewed in Section 2.3.

2.1 Meta-Heuristics

The term *meta-heuristic*, introduced by Glover (1986), derives from the composition of Greek εὑρίσκω (heuriskō, to find) and a suffix μετά (meta, behind or in an upper level). Although this term has been widely adopted, there is still not an extensively accepted definition. Some definitions from previous researchers are quoted as follows. It can be synthesized that a *meta-heuristic* usually combines the basic heuristics (such as first-come, first-served) and dynamically controls searching by feedback in a problem solving.

> A meta-heuristic is a heuristic method within which another heuristic local search procedure can be used at each step. (de Werra & Hertz, 1989)

> A meta-heuristic is a high-level and problem-independent definition of a search strategy that can then be specialized to the specific optimization problem at hand. (Birattari, 2009, p. 28)

> Meta-heuristics are typically high-level strategies which guide an underlying, more problem specific heuristics, to increase their performance. ...Many of the methods can be interpreted as introducing a bias such that high quality solutions are produced quickly. This bias can be of various forms and can be cast as descent bias (based on the objective function), memory bias (based on previously made decisions) or experience bias (based on prior performance). (Stützle, 1998)

2.1.1 Well-known algorithms

As a subclass of heuristics, meta-heuristics cannot guarantee finding the optimal solutions. However, various meta-heuristic approaches have been widely adopted and can be found in thousands[16] of published works. Some early meta-heuristics were developed decades before the proposal of the term itself, such as the local search (Croes, 1958; Lin, 1965). Comprehensive surveys of meta-heuristics can be found in C. Blum and Roli (2003) and Gendreau and Potvin (2010). Some well-known meta-heuristics (in chronological order) are listed as follows.

Local search (or hill-climbing) starts from a feasible solution and repeatedly replaces it with a better *neighbor*, until no better neighboring solutions can be found (Croes, 1958; Lin, 1965). Successors of local search include *iterated local search* (Lourenço, Martin, & Stützle, 2003), *variable-depth search* (van der Bruggen, Lenstra, & Schuur, 1993) and *variable neighborhood search* (Mladenović & Hansen, 1997).

Simulated annealing solves problems in a way similar to *annealing* used by metallurgists (Kirkpatrick, Gelatt, & Vecchi, 1983; Černý, 1985).

Genetic algorithm is a population-based meta-heuristic with repeated *mutation*, *crossover*, and *selection* (Holland, 1992). The genetic algorithm and closely related genetic programming (Koza, 1992, pp. 79–120), *artificial immune system* (Hunt & Cooke, 1996) and *Memetic algorithm* (Moscato &

[16] Google Scholar™ (http://scholar.google.com/) returned 1,200 results with "metaheuristics" in title, 1,130 for "metaheuristic", 400 for "meta-heuristics" and 587 for "meta-heuristic", on April 23rd, 2012.

Cotta, 2003) are also called *evolutionary algorithms* (Eiben & J. E. Smith, 2003, pp. 15–35).

Tabu search employs a *short-term memory* to prevent recent moves, an *intensification* mechanism for focused explorations, and a *diversification* mechanism for search balance (Glover, 1986; Glover & Laguna, 1997).

Greedy randomized adaptive search procedures (GRASP) begin with multiple heuristic (greedy random) initializations (Feo & Resende, 1989, 1995).

Ant colony optimization holds a positive feedback mechanism based on the density of "pheromone" and a "random proportional transition rule" (Colorni, Dorigo, & Maniezzo, 1992; Bonabeau, Dorigo, & Theraulaz, 1999).

Particle swarm optimization evolves from the collective behavior of an animal school (Kennedy & Eberhart, 1995; Kennedy, Eberhart, & Shi, 2001).

Estimation of distribution algorithms (EDAs) estimate the relations from the preceding population and manipulating population evolution according to the relations involved (Mühlenbein & Paaß, 1996; Mühlenbein, Bendisch, & Voigt, 1996).

Cross entropy method is a general *Monte Carlo* approach initially developed by Rubinstein (1997) for estimating the probability of rare events in complex stochastic networks, and adopted for solving combinatorial optimization problems later (Rubinstein, 1999).

As two well-known meta-heuristics which are closely related to the algorithm in this thesis, EDAs and ant colony optimization usually intends to model the variables as independent values or values with very low correlations. Some neighborhoods of local search consist of a certain amount of neighboring solutions with a limited number of variables to be flipped (for binary ones) or changed. The neighborhoods in this form implicitly refer to low correlations between variables. In fact, local search methods on such neighborhoods *can* be highly competitive in many problem domains. For example, the Lin-Kernighan-Helsgaun (LKH) (Helsgaun, 2000, 2009) which is an iterated k-Opt ($k \leq 8$ usually) local search kept most of the records of open benchmark instances[17] for TSPs. At the same time, the greedy heuristics, such as Bentley (1992)'s *multiple fragment merging* greedy, could find a solution within about 15% over HK bound very quickly for a TSP, see DIMACS TSP Challenge. The idea of greedy construction heuristics, i.e., solving a problem by assigning each variable with a limited number of values greedily and independently, is also based on the hypothesis of the independency of variables. The effectiveness of such neighborhoods (and greedy heuristics) seems to undermine, at least in certain NP-Complete problems, the hypothesis of high correlations among variables.

2.1.2 Sampling in meta-heuristics

There are a lot of explicit or implicit sampling methods in many meta-heuristic approaches, such as the evolutionary algorithms, ant colony op-

[17] See http://www.tsp.gatech.edu/vlsi/summary.html for VLSI TSP Data Set. See http://www.akira.ruc.dk/~keld/research/LKH/DIMACS_results.html for random and clustered instances in DIMACS TSP Challenge.

timization, particle swarm optimization, GRASP, EDAs and cross entropy method. A typical example is the EDAs, also known as *probabilistic model-building genetic algorithms*. EDAs were originally designed as an alternative for evolutionary algorithms (Mühlenbein & Paaß, 1996; Mühlenbein, Bendisch, & Voigt, 1996). However, EDAs inherit neither crossover nor mutation operators. Instead, the essence of the method is in estimating the relations from the preceding sample and manipulating sample evolution.

Take the *OneMax* problem (maximizing the Manhattan norm of a binary vector $\vec{x}$) as an example:

$$\max \quad \|\vec{x}\|_1 = \sum_{i=1}^{n} |x_i| \tag{2.1}$$

$$\text{s.t.} \quad x_i \in \{0,1\} \qquad\qquad 1 \leq i \leq n \tag{2.2}$$

The first stage consists in generating an initial sample S_0 of a number (non-zero) of individuals by the probability distribution:

$$P_0(\vec{x}) = \prod_{i=1}^{n} P_0(x_i), \tag{2.3}$$

where $P_0(x_i)$ is the initial probability of $x_i = 1$. The second stage consists in selecting "good" (with higher objective values) individuals to form a new sample $S_0^{\text{good}} \subseteq S_0$. In the third stage, the probability distribution is re-estimated:

$$P_1(\vec{x}) = \prod_{i=1}^{n} P(x_i | S_0^{\text{good}}). \tag{2.4}$$

The three steps can loop until a stop criterion is met. Products in Equations 2.3 and 2.4, an implicit local search-like diversification, require that each

variable is independent of the others. For those cases in which variables are dependent on others, Mühlenbein and Mahnig (1999) proposed the *factorized distribution algorithm* and Pelikan, Goldberg, and Cantú-Paz (1999) developed the *Bayesian optimization algorithm* driven by Bayesian networks. Both of these two algorithms and their successors inherited the probability-assignments sampling essence of EDAs.

However, the impact of EDAs in solving NP-Complete combinatorial optimization problems seemed limited. For example, Ceberio, Irurozki, Mendiburu, and Lozano (2012) showed that the Edge Histogram Based Sampling Algorithm (EHBSA), one of the best EDAs, could find satisfactory solutions (0.37% above optima on average) for very small ($n < 150$) benchmark instances of TSP in $100n$ iterations. Tsutsui, Pelikan, and Goldberg (2003) incorporated a 2-Opt local search to EHBSA and showed a very high chance to return an optimal solution in tens of thousands of iterations for a TSP with $n = 439$ cities, under certain configurations of parameters. Nevertheless, the state-of-the-art EDAs seemed not able to compete with other non-dominated heuristics. The main difficulty when applying the EDAs is the estimation of the probability distribution. EDAs also have at least four parameters to be determined: population size, selection size, offspring size and selection method.

In the ant colony optimization, Colorni et al. (1992)'s *roaming ant* was another algorithmic concept close to the *individual* in EDAs. Ant colony algorithms employ similar probability-assignments sampling, and some different probability updating rules as their main mechanisms. A basic ant colony algorithm called *ant system* was proposed by Colorni et al. (1992) for

solving TSPs. In each iteration, a number of ants traverse the graph and build complete paths of n edges. For an ant at a city c_i, the probability of choosing a path (c_i, c_j), where $i \neq j$, is according to the "random proportional transition rule" (Bonabeau et al., 1999, pp. 42–43):

$$P_{ij} = \begin{cases} \dfrac{(\tau_{ij})^{\alpha} \cdot (d(c_i, c_j))^{-\beta}}{\sum_{l \text{ not visited}} (\tau_{il})^{\alpha} \cdot (d(c_i, c_l))^{-\beta}} & \text{if } c_j \text{ was not visited} \quad (2.5a) \\ 0 & \text{otherwise,} \quad (2.5b) \end{cases}$$

where α and $-\beta$ are two parameters controlling the importance of the trail *intensity* (τ_{ij}, quantity of pheromone) and the *distance*[18], respectively. It can be found that the intensity τ bears a resemblance to the probability P in EDAs. The difference is that the nearness is also considered in the sampling. Hence the sampling is closer to the nearness-based neighborhoods of a local search. A selected ant leaves a certain amount of pheromone $\Delta\tau_{ij}$ on its entire course after each iteration, the quantity of which depends on the *quality* of the solution found by the ant:

$$\Delta\tau_{ij} = \begin{cases} \dfrac{Q}{\textit{Tour length found by the ant}} & \text{if the ant chose } (c_i, c_j) \quad (2.6a) \\ 0 & \text{otherwise,} \quad (2.6b) \end{cases}$$

where Q a fixed parameter. A "forgetting-bad-solution" mechanism, *evaporation* with a factor ρ, was also introduced to update the τ_{ij} to:

$$(1 - \rho)\tau_{ij} + \sum_{\text{all ants}} \Delta\tau_{ij}. \quad (2.7)$$

There were also a number of derivations of ant colony approaches. The

[18] $(d(c_i, c_j))^{-1}$ is also known as the *visibility*.

ant system in Dorigo, Maniezzo, and Colorni (1991) employed an *elitist ants* mechanism, in which the best ant (that which traversed the shortest path) deposits a large quantity of pheromone, with a view to increase the probability of the other ants of exploring the most promising solution. In Gambardella and Dorigo (1995)'s *Ant-Q*, the rule of local update was the *Q-learning*, a reinforced learning. However, no improvement compared with the ant system could be demonstrated. Besides, even in the opinion of the authors, this algorithm is not more than a pre-version of the *ant colony system*. Dorigo and Gambardella (1997a, 1997b) presented the ant colony system, and later with a 3-Opt local search, to improve the performance for problems of higher dimensions, by introducing a few mechanisms:

- A parameter $q_0 \in (0,1)$ for a balance between diversification and intensification for each ant at a city c_i:

$$c_{\text{next}} = \begin{cases} \arg\max_{c_j \in C, i \neq j} \left(P_{ij} \text{ by Equation 2.5a} \right) & \text{if } q \leq q_0 \quad (2.8a) \\ \text{the city chosen in the usual way} & \text{otherwise} \quad (2.8b) \end{cases}$$

  where $q \in [0,1]$ is a uniformly distributed random variable,

- A global update procedure, in which only the best ant would do a pheromone update in the global update,

- A list of candidates of the closest cities, which was partly inherited from local search.

In fact, the first mechanism within the ant colony system brought more intensification (focusing on the most "promising" candidate) into the sampling.

Stützle and Hoos (1997) proposed a *MAX-MIN ant system* based on the ant system and presented some notable differences:

- The values of the trails are limited by $\tau_{\min}$ and $\tau_{\max}$;
- Only the best ant updates a trail of pheromone;
- The trails are initialized with the maximum value $\tau_{\max}$;
- The updating of the trails is made in a proportional manner, the strongest trails being less reinforced than the weakest;
- A re-initialization of the trails can be carried out.

The final results are obtained by updating the best-so-far solution with an increasingly strong *frequency*, during the running of the algorithm. The first mechanism within the MAX-MIN ant system, compared to that of the ant system, made more diversifications in the sampling.

However, the experiments of ant colony optimization approaches only showed a limited impact in solving NP-Complete combinatorial optimization problems. Early approaches without local search could be easily dominated (Dorigo, Maniezzo, & Colorni, 1991), even in small-scale benchmark instances. For example, the results of ant colony methods for TSPs were usually around 0.3-2% (Stützle & Hoos, 1997) and were limited within $n < 10^3$ due to (relatively) expensive tour construction and a great number of iterations. Some later ant colony approaches employing local search methods could improve and solve TSP instances beyond 10^3 cities, such as in Dorigo and Gambardella (1997a) and L. Li, Ju, and Zhang (2008).

Another typical example was GRASP. The multi-start of GRASP, usually greedy randomized solutions, can be regarded as a biased Monte Carlo sampling. One difference between the multi-start of GRASP and the probabilistic selection in EDAs and ant colony approaches is that there is no learning or evolving for the sampling. Feo and Resende (1995) presented an example of GRASP with random selection on greedy subsets of assignments. A restricted candidate list was defined for each variable to exclude the unpromising candidates and a component-by-component construction heuristic chose solution components randomly from the restricted candidate lists to form a solution.

Another example was a probabilistic greedy in a multi-agent implementation of GRASP called Dynamic Scheduling Agents with Federation Organization (DSAFO) for an airport ground service scheduling problem at Beijing Capital International Airport (Fan & Xue, 2006; Xue & Fan, 2007). In DSAFO, resources such as baggage trucks were divided into groups. A representative agent was designed to manage a group of resources, to query tasks from the *blackboard* and to try to assign the published task with its own resource according to the *earliest finish time* heuristic. The tasks were listed one by one on the "blackboard" according to an *earliest due date* heuristic. The only randomness was a priority of communication for each representative agent. Some researchers also embedded a local search in the GRASP to improve the results further (Feo & Resende, 1995; Marinakis, Migdalas, & Pardalos, 2005b).

GRASP is flexible and easy to implement. The results of GRASP can be generally competitive with basic heuristics, such as the DSAFO outperformed

the earliest due date heuristic, the earliest ready time heuristic and a MAX-MIN ant system in the bi-objective optimization of airport ground service scheduling. However, the non-dominance was not always guaranteed. For example, Marinakis et al. (2005b) showed a comparison on solution quality between an iterated local search, involving GRASP and a 3-Opt local search, and other fifty four algorithms submitted in the DIMACS TSP Challenge in some benchmark TSP instances ($10^3 \leq n \leq 2.4 \times 10^3$). Marinakis et al. (2005b)'s method generally returned an average 1.181% excess over optima and was ranked as the thirteenth. However, their result was still not able to be competitive with other iterated 3-Opt local search methods such as D. S. Johnson and McGeoch (1997)'s (with an average 0.413% excess). In addition, the time cost was not reported in Marinakis et al. (2005b). Marinakis, Migdalas, and Pardalos, 2005a presented another GRASP approach and showed a twelfth rank.

Differing from the methods mentioned above, many other meta-heuristics that employ sampling treat one solution as an individual and the whole search space, or a promising part of it, as the population for a given problem. Different methods were developed to estimate the promising (hopefully suboptimal) solutions by investigating the promising subset in the sample. A number of new algorithms have been developed, and a lot of interesting findings have been achieved to guide and to control the heuristic search procedure. The resulting meta-heuristics returned, in general, more satisfactory results than conventional simple heuristics. However, for a certain problem domain, the *ad-hoc* methods such as a local search algorithm with a sophisticated and well-designed neighborhood usually were more com-

petitive with the sampling-based methods, especially in large-scale problem instances. A possible reason was the distributions of suboptima (or local minima/maxima) in an NP-Complete problem were extremely irregular and difficult to estimate. In addition, the decision procedure of the optimal sampling plan is, ironically, another problem to optimize, for example, Raschke, Krishen, Kachroo, and Maheshwari (2012) discussed a combinatorial optimization model for identification and optimization of sampling in certain conditions.

2.2 Hyper-Heuristics

A *hyper-heuristic* is a search method or learning mechanism for selecting or generating heuristics to solve computational search problems (Burke, Hyde, Kendall, Ochoa, Ozcan, & Qu, 2010). The term itself was introduced by Denzinger, Fuchs, and Fuchs (1997) and the definition became well-accepted by Burke, Kendall, et al. (2003). The prefix was from ὑπέρ (hyper, above or with many dimensions). In fact, the relation between hyper-heuristics and meta-heuristics, i.e. "hyper-heuristics versus meta-heuristics", is not very clear due to the different definitions of meta-heuristics. For example, hyper-heuristics $\not\subseteq$ meta-heuristics could be held in de Werra and Hertz (1989)'s definition, as quoted in Section 2.1. However, hyper-heuristics $\subseteq$ meta-heuristics, which could be held in other definitions such as Stützle (1998)'s, is assumed in this thesis, as shown in Figure 1.7. What distinguishes hyper-heuristics from the meta-heuristics is the search space. A hyper-heuristic generally solves a problem by calling given or generated heuristics, which are called low-level heuristics (LLHs), and does not directly manipulate the

assignments of variables within a solution. Therefore, the search space of a hyper-heuristic is a space of combinations of LLHs. On the other hand, a meta-heuristic *can* directly manipulate the meta-data in a solution. The search space of a meta-heuristic *can* be a space of solutions.

Because of the "off-the-peg" (Burke, Hyde, Kendall, Ochoa, Ozcan, & Qu, 2010) nature, hyper-heuristics are generally easier to be adapted to different problem domains than meta-heuristics. As a side effect accompanying the high adaptability, the resulting selected (and some generated) heuristics, in many cases, were dominated by those meta-heuristics carefully fine-tuned for a problem domain. However, some hyper-heuristics (mostly heuristic generation) could be *non-dominated* and outperformed existing heuristics such as Chan et al. (2012)'s PHunter hyper-heuristic, as mentioned in Section 1.3. It was believed that one of the main contributors to the new upper bounds found by the PHunter was a newly introduced LLH which had not been implemented before or carefully fine-tuned in any other approaches. If it was proved true, a heuristic generation methodology which improved existing LLHs could notably increase the competitiveness of a general hyper-heuristic.

Burke, Hyde, Kendall, Ochoa, Ozcan, and Qu (2010) surveyed comprehensive hyper-heuristic approaches and classified all the hyper-heuristics into two categories: heuristic selection and heuristic generation. In this thesis, the two categories are formally redefined according to the countability of the maximum space of LLHs, as shown in Section 3.2.5. As a result, both extensions of heuristic selection and heuristic generation are extended. Because Burke, Hyde, Kendall, Ochoa, Ozcan, and Qu (2010)'s review was so

general and comprehensive, it is not necessary to repeat every well-known hyper-heuristic approach in this thesis. The approaches that are critically important, newly developed or missed in Burke, Hyde, Kendall, Ochoa, Ozcan, and Qu (2010) are introduced in the following.

2.2.1 Heuristic selection approaches

The algorithm selection problem was first formalized by Rice (1976). The question in an algorithm selection problem (Xu, Hutter, Hoos, & Leyton-Brown, 2008) is: "which algorithm(s) should run now in order to minimize some performance objective, such as the expected runtime?" Although the resulting selected algorithm seems more effective than an arbitrary algorithm on average, the algorithm selection problem, itself, is still governed by Theorem 1 of NFL if it is regarded as a BBO, which means that no specific information is considered. In fact, determination of the best algorithm to solve one algorithm selection problem is, ironically, another algorithm selection problem (Smith-Miles, 2008a).

Early hyper-heuristic approaches, similar to the early meta-heuristics, were proposed far earlier than the term. For example, Crowston, Glover, Thompson, and Trawick (1963) sampled a set of decision rules and updated the probabilities of the rules to learn (or converge) to improve the search process. Crowston et al. (1963) showed the quality of the solution found by learning was more satisfactory than those by single isolated rules or by random sampling among the rules. That was also one of the early heuristic selection applications.

Burke, Hyde, Kendall, Ochoa, Ozcan, and Qu (2010)'s heuristic selection

concept referred to the methodologies for choosing or selecting existing heuristics. In this thesis, heuristic selection is defined as the hyper-heuristics of which the possible results (space) of LLHs are countable for any problem, including those with countably infinite variables. Therefore, some research areas used to be "close" to but outside the heuristic selection in Burke, Hyde, Kendall, Ochoa, Ozcan, and Qu (2010) are regarded as heuristic selection methods in this thesis, such as the parameter optimization approaches for heuristics and the (heuristic) algorithm portfolios. See Section 3.2.5 for more details.

Tuning parameter set automatically for heuristics could be considered as early practice in hyper-heuristics. An early example was the use of Rechenberg's 1/5 success rule for changing the mutation rate by evolution strategies in an evolutionary algorithm presented in 1973 (Burke, Hyde, Kendall, Ochoa, Ozcan, & Qu, 2010). Some simple heuristics as well as many meta-heuristics have a number of parameters to be determined in each problem domain. Population-based meta-heuristics received much attention, such as Grefenstette (1986), Eiben, Hinterding, and Michalewicz (1999) and Lobo and Goldberg (2004)'s different methods of controlling parameters for evolutionary algorithms, respectively. For local search heuristics, an example was Boyan (1998) and later Boyan and A. Moore (2000)'s parameter optimization method "STAGE" in seven problem domains, including bin-packing and the Boolean Satisfiability Problem (SAT). STAGE employed a fixed number of experience-based attributes of the current solution to estimate the performance of each given local search heuristics by some linear, quadratic, cubic and other regression methods. The best dynamically estimated local

search won the turn to run. STAGE was proven effective and efficient and could be compared to up-to-date heuristics, such as Kim and Wy (2010)'s results in 1D bin packing. Many recent advances of hyper-heuristics, such as Mısır, Verbeeck, Causmaecker, and Vanden Berghe (2012)'s approach, which won the CHeSC 2011 competition, had involved parameter optimization for manipulating LLHs.

A *portfolio of algorithms* is a collection of different algorithms and/or different copies of the same algorithm running on different processors (Gomes & Selman, 1997; Huberman, Lukose, & Hogg, 1997), or equivalently on fractions of CPU cycles on a single CPU. The first solution (for satisfiability-style problems) or the best solution (for minimization or maximization problems) is chosen as the result of the portfolio. Xu, Hutter, et al. (2008)'s portfolio of solvers, SATzilla, which won three gold medals, one silver medal and one bronze medal in the 2007 SAT competition[19], is a typical example. SATzilla predicts performances (such as computation time) of solvers by tens of features (or attributes) and determines the best subset of solvers to run. Chan et al. (2012)'s PHunter was another example. Given a specific problem, PHunter tests all given LLHs in crossover, mutation and ruin-and-recreate diversification classes, and constructs three portfolios with the best three LLHs in each class, respectively. The iterations of the three LLHs are 3, 2 and 1, respectively, for each portfolio. The final combination of portfolios was determined by pre-trained decision tree prediction according to tens of attributes. In addition, parallel execution of heuristic and exact algorithms are also closely related to heuristic algorithm portfolios, such as Denzinger

[19]See http://www.satcompetition.org/2007/.

and Offermann (1999)'s multi-agent based approach for achieving cooperation between different search-systems and Talukdar, Baerentzen, Gove, and De Souza (1998)'s asynchronous team of cooperative agents.

Some hyper-heuristics returned previously unknown heuristics, and hence might look like heuristic generation methods. However, many of them are actually heuristic selection methods, when the spaces of LLHs are countable or even finite for an input problem with countably infinite variables. Even though the result may seem to be new, powerful and difficult to discover for experienced human managers, such a hyper-heuristic is equivalent to a selection process on ready-to-use LLHs by a one-to-one transformation. An early example was Bernstein (1987)'s method "SH". SH builds a population of new heuristics from a carefully chosen vocabulary, collects statistics on their effectiveness, identifies those that work very well, or those most likely to improve a bad schedule radically. It is not hard to find that SH was a *populate-and-select* style heuristic selection method, while the space of possible algorithms was limited by a constant number. Vázquez-Rodríguez and Petrovic (2010)'s method searches simultaneously for the best sequence of 13 dispatching rules and the number of operations to be handled by each dispatching rule. Vázquez-Rodríguez and Petrovic (2010)'s approach was equivalent to setting up a priority parameter belonging to $\{0, 1, \ldots, 13\}$[20] to each dispatching rule and a positive integer to a pseudo parameter "length". Other instance-specific selection examples that employed machine learning techniques include Koonce and Tsai (2000)'s decision-rule-like heuristic for dispatching priority in production scheduling, Kwak and Yih (2004)'s

[20] Where 0 stands for not selected.

discovery of the decision tree approach (competitive decision selector) in production control, X. Li and Olafsson (2005) and later Olafsson and X. Li (2010)'s decision tree for dispatching jobs, Burke, Petrovic, and Qu (2006)'s cased-based reasoning heuristic selection method in timetabling problems and Xue, Chan, et al. (2011)'s class association rule learning in Euclidean TSPs. There also were data set-specific (or family-specific) examples, such as Lee, Piramuthu, and Tsai (1997)'s dispatching rule discovery embedded in a genetic algorithm and Sha and C.-H. Liu (2005)'s discovery of a new due date assignment heuristic (as a decision tree) for the dynamic job shop problem, respectively.

In many heuristic selection methods, learning was regarded as the essence of hyper-heuristics. That was especially obvious in the heuristic discoveries mentioned above, where machine learning techniques were generally involved (Koonce & Tsai, 2000; Kwak & Yih, 2004; Sha & C.-H. Liu, 2005; X. Li & Olafsson, 2005; Olafsson & X. Li, 2010). These methods revealed a promising class of applications of heuristic selection (or generation) — tremendous machine learning algorithms in certain conditions, as shown in Section 2.3.

2.2.2 Heuristic generation approaches

Burke, Hyde, Kendall, Ochoa, Ozcan, and Qu (2010) described heuristic generation as "generating new heuristics from the components of existing ones". However, some heuristic discovery approaches are precisely equivalent to other heuristic selection processes, such as Bernstein (1987)'s populate-and-select method shown in Section 2.2.1. Such an algorithm could probably possibly be misclassified by Burke, Hyde, Kendall, Ochoa, Ozcan, and Qu

(2010)'s classification. This section presents a number of typical heuristic generation approaches, which are so sophisticated and may actually return one or many LLHs out of an uncountable number of possible LLHs.

A typical example of heuristic generation is Joslin and Clements (1999)'s "*squeaky wheel*" *optimization*. The squeaky wheel implements an order of priority for building components such as the variables waiting for assignments. There is a certain construction process, such as a greedy heuristic, that builds a solution according to a given priority ordering. Trouble spots, "bottleneck" components or the key components to change for a better solution, are identified through analyzing the constructed solution. Then the priority ordering is updated by giving the trouble spots higher priorities and the next iteration starts unless the termination condition is met. The LLH space of "squeaky wheel" optimization is bounded in $\Theta(n!)$ generally, where n is the number of the components (variables usually). However, only a portion of the space can be reached in practice if the number of the variables is great enough. Aickelin, Burke, and Li (2009) presented an updated version, evolutionary "squeaky wheel" optimization, in which selection and mutation operators were involved between analysis of the solution and prioritization of priority ordering.

Another example is genetic programming (Koza, 1992, pp. 79–120). Genetic programming is an evolutionary algorithm-based methodology. The goal of genetic programming is evolving computer programs in a natural selection way for certain problems. Particularly, the chromosomes in general evolutionary algorithms are programs. For example, there are two equations "$3 \times x + 4$" and "$1 - 2 \div y$" selected from a population in a certain iteration.

The results of a mutation operation may be "$3 \times x + 5$" and "$1 - 1 \div y$", respectively. The results of a crossover operator may be "$3 \times x + 2 \div y$" and "$1 - 4$". The size of the space of possible programs is bounded in $\Theta(2^n)$, if there are two free "characters" in the vocabulary and the equation is a 1D string, where n is the maximum size of the program. The size of the space will increase dramatically, if there are more characters and more complex forms of equations. Therefore, in practice, some constraints like the "maximum size of program" are usually introduced to restrict the space of LLHs. It is not necessary for every genetic programming to be a heuristic generation, because there are the programs that can be seen as solutions directly. Examples of heuristic generation of genetic programming approaches can be found in Ho and Tay (2005), Jakobović, Jelenković, and Budin (2007) and Tay and Ho (2008) in production scheduling, Burke, Hyde, and Kendall (2006) and Burke, Hyde, Kendall, and Woodward (2007) in bin packing, Fukunaga (2008) in SAT, Keller and Poli (2008) in TSP, and Pillay and Banzhaf, 2007 and Bader El Den and Poli (2009) in timetabling.

2.3 Supervised Learning

Supervised learning, supervised machine learning, or classification, is the search for algorithms that reason from externally supplied records (or instances) to produce general hypotheses, which then make predictions about future instances (Kotsiantis, 2007). In other words, supervised learning aims at building a concise model of the distribution of class labels in terms of predictor features (or decision attributes) (Kotsiantis et al., 2006). An attribute can be continuous, binary or categorical, but a label usually accepts discrete

Table 2.1: Examples of records with known labels

Case	*T'ien*'s horse (attribute 1)	King's horse (attribute 2)	Date (attribute 3)	Was *T'ien*'s faster? (label)
1	high-grade	high-grade	day 1	no
2	high-grade	high-grade	day 2	yes
3	high-grade	high-grade	day 3	no
4	high-grade	middle-grade	day 1	yes
5	high-grade	middle-grade	day 2	yes
6	high-grade	middle-grade	day 3	yes
...	...	...	...	...

or categorical values only. Table 2.1 shows some examples of instances with known labels from *T'ien*'s possible records of horse-racing. Supervised learning is considered as a classification problem due to the discrete labels and the result of a supervised learning is a classifier. The results are usually classification rules, regression functions or known examples. A possible classification rule learned from the data in Table 2.1 is "if *T'ien*'s horse is high-grade and the king's is middle-grade then *T'ien*'s horse is faster."

2.3.1 Well-known classification techniques

A taxonomy of some well-known classification techniques can be organized as a tree, as shown in Figure 2.1. Classification methods can be divided into more and more specialized and well formalized methods such as those on the right side of Figure 2.1. The discriminant function methods, for instance, subsume recursive partitioning, which further includes three algorithms: LTREE, CART and C4.5.

Three of the most efficient methods among those listed in Figure 2.1, Naïve Bayes normal, C4.5 and Ripper, are employed in this thesis. Therefore,

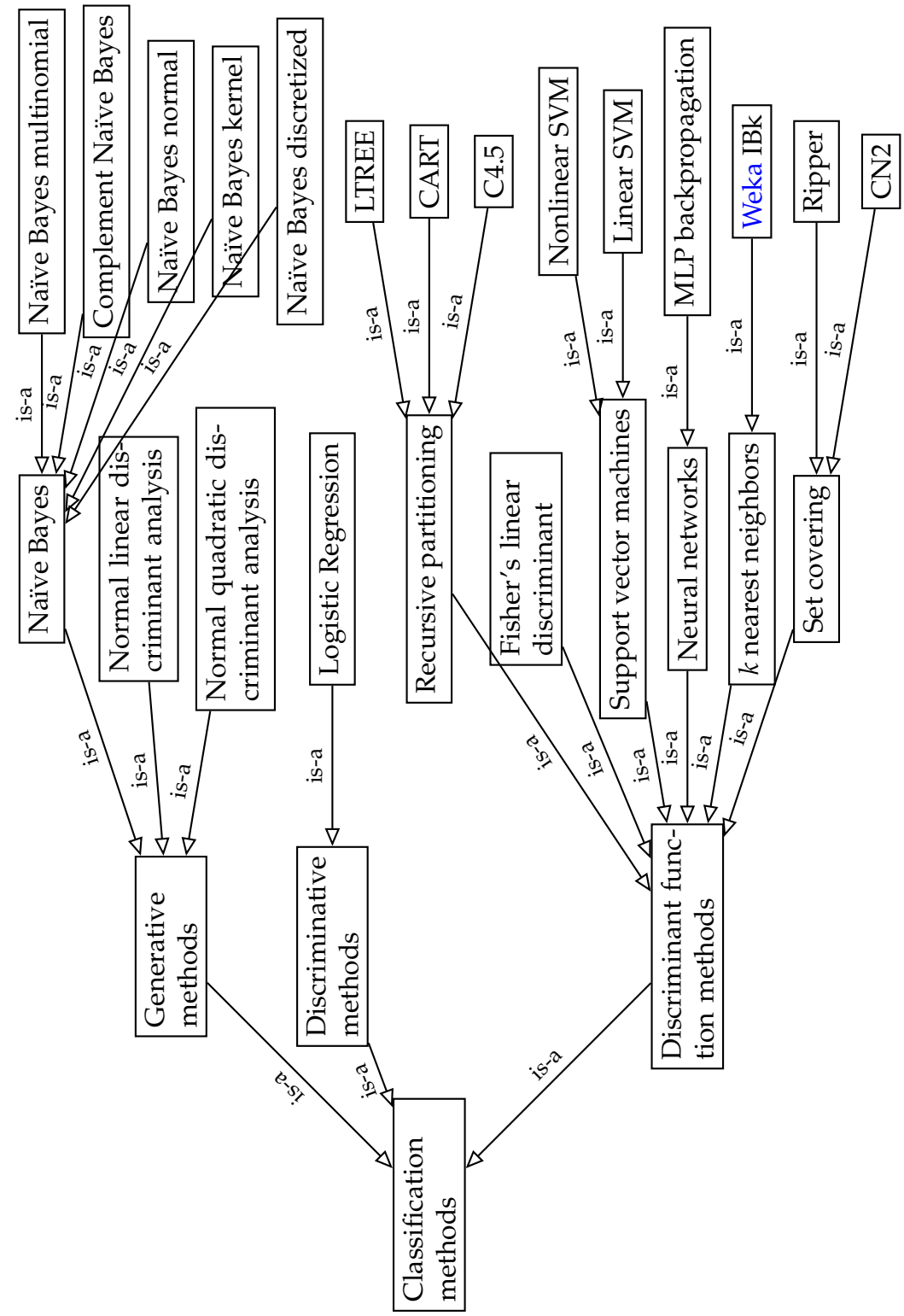


Figure 2.1: A tree taxonomy of some well-known classification techniques (Hilario, Kalousis, Nguyen, & Woznica, 2009)

Table 2.2: References for further reading on the classification techniques not selected in this thesis

Classification method	Reference
Naïve Bayes multinomial	Rennie, Shih, Teevan, and Karger (2003)
Complement Naïve Bayes	McCallum and Nigam (1998)
Naïve Bayes kernel	John and Langley (1995)
Naïve Bayes discretized	John and Langley (1995)
Normal LDA	Lachenbruch and Goldstein (1979)
Normal QDA	Lachenbruch and Goldstein (1979)
Logistic Regression	Le Cessie and Van Houwelinge (1992)
Fisher's LD	Fisher (1936)
LTREE	Gama (1999)
CART	Breiman, Friedman, Olshen, and Stone (1983)
Support vector machines	Suykens and Vandewalle (1999)
Neural networks	G. P. Zhang (2000)
Weka IBk	Aha, Kibler, and Albert (1991)
CN2	Clark and Niblett (1989)

they are introduced in detail below. There are two criteria to select classification techniques to use. One criterion is the efficiency (or quick running) of each algorithm. Another is the overall diversity, i.e., to choose from different categories. Suggested literature for further reading on other classification techniques is listed in Table 2.2.

Naïve Bayes normal is a probabilistic classifier based on Bayes' theorem

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}. \tag{2.9}$$

In addition, all attributes (variables) are considered independent (naive) of each other. Although the assumption of independency seems poor and naive, Naïve Bayes usually competes well with other sophisticated methods (Rish, 2001). Given attributes $\vec{a} = \{a_1, a_2, \ldots, a_n\}$ and a class C, the likelihood of C is

$$P(\vec{a}|C) = P(a_1, a_2, \ldots, a_n|C) = \prod_{i=1}^{n} P(a_i|C), \tag{2.10}$$

1. Check for base cases
2. For each attribute a
   2.1. Find the feature that best divides the training data such as information gain from splitting on a
3. Let a_{best} be the attribute with the highest normalized information gain
4. Create a decision node *node* that splits on a_{best}
5. Recurse on the sub-lists obtained by splitting on a_{best} and add those nodes as children of *node*

Figure 2.2: Pseudo codes of the C4.5 classifier

because of the assumption of independency. "Normal" means each attribute is estimated like a normal distribution, not in a kernel estimator way (John & Langley, 1995) and there is no supervised discretization to convert numeric attributes to categorical values. Examples of successful applications of the Naïve Bayes normal include text classification (McCallum & Nigam, 1998) and bio-informatics (Yousef et al., 2006). An open source Java implementation of Naïve Bayes normal provided in Weka (See Section 4.1) is employed, as described in Section 4.3.

C4.5 generates a decision tree based on the concept of *information entropy* (Quinlan, 1993, pp. 17–24). Kotsiantis (2007) summarized the pseudo codes of C4.5 concisely, as shown in Figure 2.2. The decision tree is created in a greedy way by iteratively maximizing the normalized information gain on each node branching. Quinlan later developed improved commercial successors[21] C5.0 (for Unix/Linux) and See5 (for Windows). This thesis employs J48, which is an open source Java implementation of C4.5 provided in Weka.

Cohen (1995) proposed the *Ripper* method, or *repeated incremental pruning to produce error reduction*. There are two main phases in Ripper. The first phase is a building phase, where two actions, a growing action and a pruning

[21] See http://www.rulequest.com/see5-info.html.

action, are repeatedly performed. In the growing, conditions (antecedents) of rules are greedily (with highest information gain) increased until 100% confidence is met. In the pruning, conditions are pruned in a certain metric. The second phase is an optimization phase, where the size of the set of built rules is reduced to fit the training records. JRip, an open source Java implementation of Ripper provided in Weka is employed in this thesis.

2.3.2 Sampling and attribute selection for supervised learning

The amount of training records may be very large in some applications of supervised learning. For example, the Network Intrusion Data used in the 1999 KDD Cup[22] was raw TCP dump data collected for nine weeks. There were 41 attributes and one label defined for 4,940,000 training records and 3,110,290 test records. The sizes of the training data and the test data were 743 MB and 430 MB, respectively. Sampling of records is usually introduced in such large-scale data sets. The goal of sampling is to minimize the sample size while maintaining the quality of learning results (H. Liu & Motoda, 2001, p. 6). Reinartz (2002) summarized the most well-known samplings as, similar to those in the EDAs, two groups:

Random sampling which randomly chooses a subset, and

Stratified sampling which selects the records of the minority class more frequently in the imbalanced data sets, so that the minority class will not be ignored due to the vanishing probability.

[22] See http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html.

Characteristics of data are also important in sampling. Typical characteristics range from simple statistics up to information on the data quality (Reinartz, 2002). The usual way to examine sampling is by comparing the differences in the indicators defined on a classifying procedure and the learning results, such as computation time, memory cost, correctness and recall, on the sampled records and on the original data set, such as in Zaki, Parthasarathy, Li, and Ogihara (1997).

Besides the reduction of records (rows) by sampling, the attributes (columns) can also reduce some classifier indicators, such as the computation time and the memory used. An *attribute selection* (or feature selection) algorithm aims at finding the optimal subset in a given set of training attributes. An evaluation function or algorithm, that approximately indicates a promising value for a proposed attribute subset, is usually attached, too. Clearly, an attribute selection may run up to $2^n - 1$ times and cost too much time to numerate all non-empty subsets of given attributes, where $n = \|\vec{a}\|$ is the number of given attributes. So a termination condition is usually developed. An example of attribute selection techniques was Hall (1999)'s *correlation-based feature selection*, which tended to choose subsets with low correlation internally and high correlation with the label.

Because the correlations between attributes are usually not zero (partially or fully dependent), some attribute manipulating methods also *generate* new attributes to try to eliminate the correlations before attribute selection, such as (Markovitch & Rosenstein, 2002)'s FICUS algorithm. This has been called feature construction or transformation (Kotsiantis, 2007). The generated attributes might lead to more concise and accurate classification and might help

people understand some new concepts. In fact, such a transformation can also be finished by *principal component analysis* or *weighted principal component analysis* (Kriegel, Kröger, Schubert, & Zimek, 2008).

In terms of computation cost, the sampling approaches in supervised learning are considerably different from those in the meta-heuristics that consider a solution as an individual record. In supervised learning, a sampling approach generally *reduces* the number of the records notably and hence the overall computation time is usually reduced remarkably. For such a meta-heuristic, a sampling approach commonly brings *extra*, sometimes tens of times more, computation cost to *prepare* the training records to identify the "good" ones. It could be synthesized that the variables are assumed highly correlated to others in such a meta-heuristic.

2.3.3 Meta-learning

If finding the optimal classifier for a set of given records could be regarded as an optimization problem, the choice of the best classification technique for finding the optimal classifier becomes an algorithm selection problem. Particularly, such a choice of the best classification technique is called *meta-learning* (Vilalta & Drissi, 2002). The prefix "meta", which means behind or in an upper level, is exactly the same as that of meta-heuristics. Vilalta and Drissi (2002) proposed a framework of meta-learning consisting of a set of "base-learners" in the lower level and a "meta-learner" in the upper level. For example, Lindner and Studer (1999) employed case-based reasoning to govern and select supervised learning algorithms, and Xu, Krzyżak, and Suen (1992) developed a method consisting of Bayesian formalism for com-

piling predictions from a number of classifiers. In fact, the "upper level" controlling processes of meta-learning, similar to those in hyper-heuristics, are not necessarily machine learning techniques.

Meta-learning is naturally closely related (Smith-Miles, 2008b) to hyper-heuristics (especially heuristic selection). For instance, Kanda, Carvalho, Hruschka, and Soares (2011) proposed a meta-learning framework, which is actually a hyper-heuristic, to select optimization algorithms for solving TSPs. Meta-attributes were defined in the problem and the efficiency of the optimization algorithms examined. The meta-attributes included the number of cities, the number of edges, the lowest, the highest and the average costs of edges, and the most frequent edge cost. Four meta-heuristics, including tabu search, GRASP, simulated annealing and genetic algorithm, were employed as LLHs in small-scale ($n \leq 100$) TSP instances. In fact, Cruz-Reyes et al. (2012) stated an on-going trend on the shifting of (heuristic) algorithm selection methods from meta-learning to hyper-heuristics.

Chapter 3

The SPOT Hyper-Heuristic

管中窺豹, 時見一斑。

See a spot when looking at a leopard through a bamboo tube.

《世說新語・方正》

(*A New Account of the Tales of the World*)

Liu, Yiqing

The quoted adage, "see a spot when looking at a leopard through a bamboo tube," actually has had two contradictory metaphorical meanings through long use:

- (Original meaning) having a limited view thus missing the big picture;
- (Transferred meaning) being able to visualize the whole animal.

The SPOT algorithm, which involves sampling proportions, may possibly encounter the contradictory issue, too. This chapter mainly presents the SPOT algorithm. Standards and indicators are also introduced to explain why and

how the presented approach can "visualize the whole" effectively instead of having "a limited view". Section 3.1 describes an overview of the SPOT. Formal definitions of important concepts are given in Section 3.2. Particularly, heuristic selection and heuristic generation are formally redefined in Section 3.2.5. The SPOT algorithm can accordingly be classified as a heuristic generation method in general. Section 3.3 presents a typical methodology for developing the SPOT hyper-heuristic, including the determination of important parameters. Finally, Section 3.4 provides some discussion on the presented algorithm and on the formal definitions.

3.1 An Overview of the SPOT Approach

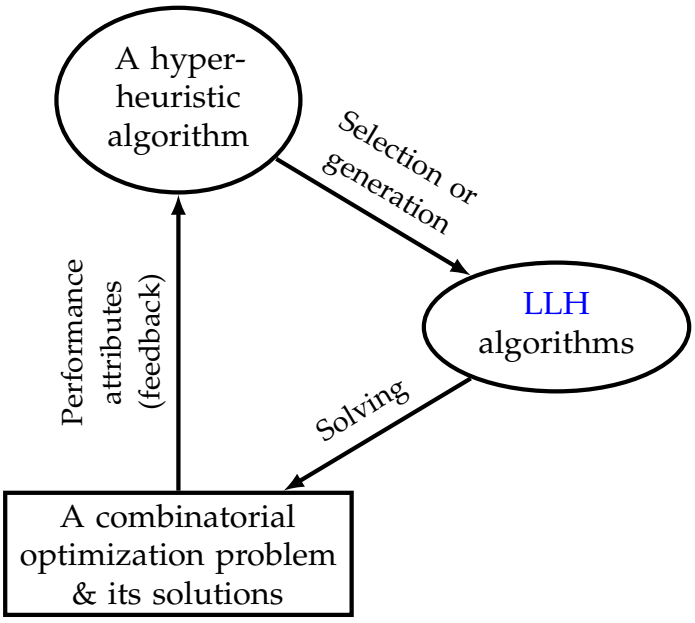
The SPOT (Suboptimum- and Proportion-based On-the-fly Training) approach is a heuristic generation methodology which modifies or creates the heuristics and/or data structures for solving combinatorial optimization problems according to the *instance-specific* information that is learned from sampled small-scale *subproblems* and their *suboptimum*[23]. The main framework of the SPOT can be described as "sample-learn-generate". The SPOT algorithm employs a standard for unconstrained problems in sampling and a number of supervised learning methods for learning instance-specific information. The attributive phrase "suboptimum-based" stands for an important idea in the SPOT — to discover knowledge about suboptima instead of optima. The attributive phrase "proportion-based" stands for another impor-

[23]In thesis, a proportion (sample) of variables is a finite set, such as 30 cities in a 3,000-city TSP; a sampled subproblem is a problem (finding a solution for some objective) defined on a sample of variables, with all the soft constraints defined on the sample, such as finding the shortest tour for the 30 cities; a suboptimum of such a subproblem is a solution not exceeding the optimum very much, such as a tour within 105% length of the optimal tour.

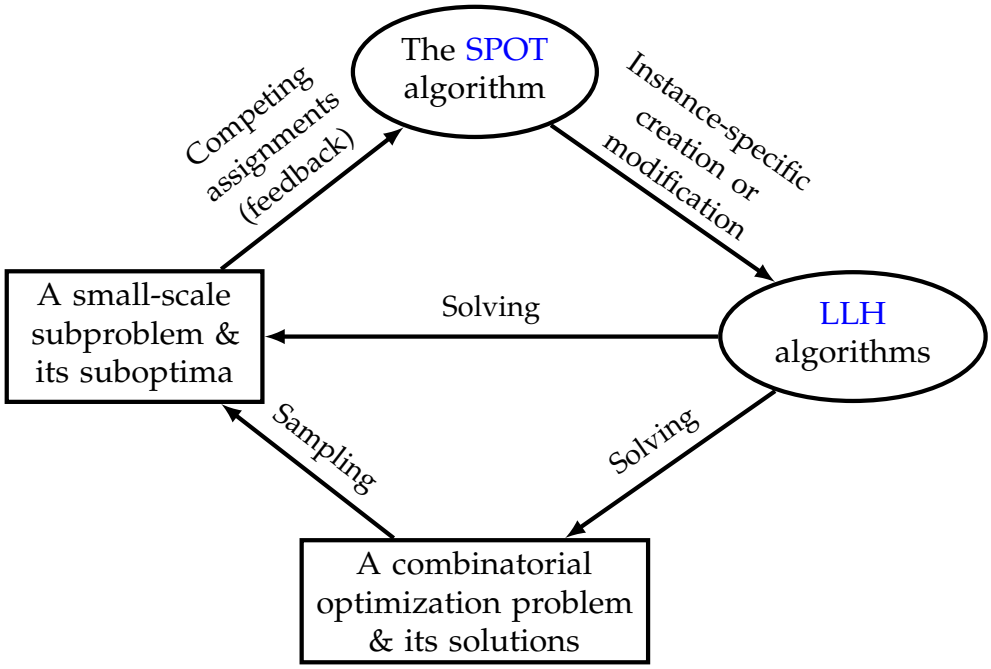
tant idea, sampling a portion of variables to form a subproblem for further investigation and learning. Based on the two ideas, the SPOT approach can estimate *all* assignments in *all* suboptima from the suboptima of some subproblems. Thus the SPOT method can be implemented as an "on-the-fly" preprocessor within the routine of a heuristic or (possibly exact) algorithm. The variables, similar to those in Naïve Bayesian methods and in EDAs, are regarded as relatively independent in the SPOT methodology.

The conceptual infrastructures of the SPOT method and a conventional hyper-heuristic are compared, as shown in Figure 3.1. There are two algorithmic objects, a set of LLHs and a hyper-heuristic algorithm, in ellipse vertices in the first figure. The data objects are shown in rectangles. The conventional hyper-heuristic algorithm dynamically controls the set of LLHs according to the feedback of some performance attributes. In comparison, the SPOT method introduces one more data object — a small-scale subproblem (and its suboptima). In the latter figure, the feedback is not directly compiled from a feasible solution to the given problem. Instead, the feedback is about the competitions among assignments of variables and collected from the suboptima of its subproblem. The SPOT algorithm creates or modifies existing LLHs to generate new LLHs according to the feedback. The generation of new LLHs is highly associated with the meta-data of the sampled subproblem and the given problem. Therefore, the SPOT algorithm is an instance-specific algorithm.

The most important concept in the SPOT methodology is the standard (or model) of the input combinatorial optimization problem, which is called U/EA (Unconstrained and with Equinumerous Assignments). A precise



(i) Framework of a conventional hyper-heuristic



(ii) The framework of SPOT

Figure 3.1: A comparison between the conceptual diagram of the SPOT methodology and that of the conventional hyper-heuristic framework

definition of the U/EA standard can be found in Section 3.2.3. The U/EA standard regulates the input problem so that there is no inter-variable hard constraints. The constraint-free condition means that any arbitrary change of any assignment in a feasible solution still results in a feasible solution. Another *assumption* is that the correlation coefficients among the variables are almost zero. The constraint-free and the correlation-free conditions enable proportion-based learning and suboptimum-based learning, as briefly described as follows.

First, some methods of solving a U/EA problem can be approximately estimated as some classifiers. In this thesis, an *assignment* (or valuation) refers to associating a variable (or a set of variables sometimes) with a certain constant value. Two assignments are called *competitors*, if their target variables are actually the same while their associated values are different. The set consisting of *all* the competitors relating to a variable is called the *group of competitor assignments* for the variable. Given a solution, there is usually one and only one assignment for each variable, thus there is one and only one "winner" assignment in each group. Such a "winner" assignment is called an *optimum assignment* if there is (at least) an optimum that contains the assignment. Henceforth, the objective of finding a solution in a search space becomes a *classification* of n optimum assignments out of n groups. In fact, the idea of transforming searching in solution space to classification or estimation of variables is not new. A typical existing example is EDAs.

Furthermore, the classifier can be approximately learned from representative subproblems (proportions) and their optima. Because of the constraint-free and the correlation-free conditions, the classification of the n optimum

assignments can then be approximately trained by a sampled *representative* subset of n' groups, like the ordinary sampling in supervised learning, where $n' \ll n$. The variables, instead of solutions, are regarded as "individuals" in the sampling, as shown in Figure 3.1. Each assignment generates one row of a training record. A training record stands for the decision attributes of a competitor assignment, and the label denotes "whether the assignment is an optimum assignment or not". The subset of groups should be representative, which is also concerned in general sampling as shown in Section 2.3.2. The optimum assignments in the n' groups are directly relating to their optima.

The classifier can be approximately learned from the suboptima of representative subproblems, too. Because of the NP-hardness, it could be too difficult to classify all the n' optimum assignments correctly and quickly for a NP-hard subproblem. On the other hand, the number of common assignments between a suboptimum and an optimum, as well as between two suboptima, should be a dominatingly large fraction due to the correlation-free condition. For instance, changing an arbitrary assignment in an optimum can probably generate a considerably large number of suboptima. Therefore, the classification label can be transformed from being optimum assignments to being *suboptimum* assignments in the n' groups. In practice, multiple subproblems can be sampled to reduce the sampling error in certain problems.

The classifier can also be approximately learned on-the-fly, if the size n' and the set of decision attributes are well restricted. The time cost of the whole learning process mainly consists of two parts: (i) the time for preparing the training records (suboptima) and (ii) the time for training the classifier. For most algorithms, the first part is highly associated with the

size n' of the subproblem. The latter part is mainly associated with the rows (number of assignments) and columns (decision attributes) for stochastic machine learning techniques. If both can be controlled to appropriate scales, the overall time cost will be well restricted to an on-the-fly level. A possible useful technique for controlling the columns is the attribute selection which speeds up learning. However, more time spent on learning usually means a more precise and more general classifier. Hence a sufficient amount of time is still necessary for learning. The balance between the precision of the classifier and the time cost should be carefully considered during the development of the SPOT hyper-heuristic. Similar decision attributes can also be found in recent advances of hyper-heuristics, such as Kanda et al. (2011)'s meta-features and the features of SATzilla (Xu, Hutter, et al., 2008).

Finally, the classifier can be transformed to heuristics. When the classifier is applied back to the given problem, each assignment can be associated with the approximate value (probability) of being a "winner" (or (sub)optimum assignment). Types of the value may include distance, weight, job demand and penalty of dissatisfactions. Then many, if not all, LLHs can be modified in their searching behavior with the values. A typical example for local search heuristics is a reordered search sequence for the (probably reconstructed) neighborhood. Some completely new LLHs might also be generated according to the values. For example, the most favorite assignment in each group is chosen to form a solution. It seems that modifying existing LLHs seems more competitive than creating completely new LLHs in practice.

3.2 Formal Definitions

This section aims at precisely describing the concepts involved in this thesis and improving the reproducibility of the presented approach. Section 3.2.1 redefines the combinatorial optimization problem, in which the extension includes the objective problem domains of the SPOT methodology. Section 3.2.2 formulates hyper-heuristics based on the concepts in the combinatorial optimization problem. The U/EA and the U/EA^2 standards are formatted in Section 3.2.3. The presented SPOT algorithm is defined in Section 3.2.4. The definitions of heuristic selection and heuristic selection are given in Section 3.2.5. Accordingly, the SPOT algorithm is regarded as a hyper-heuristic and a heuristic generation method in general.

3.2.1 Combinatorial optimization problem

The SPOT method aims at solving different domains of combinatorial optimization problems. A formal definition is extended from C. Blum and Roli (2003) in this thesis and can be given as follows.

Definition 1 (Combinatorial optimization problem)**.** *A combinatorial optimization problem* (S, f) *can be defined by :*

- *a countable set of* variables $X = \{x_1, x_2, \ldots, x_n\}$, $n \in \mathbb{Z}^+$;
- *discrete variable* domains $D_1, D_2, \ldots, D_n$;
- *a set of* possible solutions $V = v_1 \times v_2 \times ... \times v_n$, *where* $v_i = \{(x_i, \nu) | \nu \in D_i\}$ *is the set of all* assignments *(valuations) for variable* x_i;

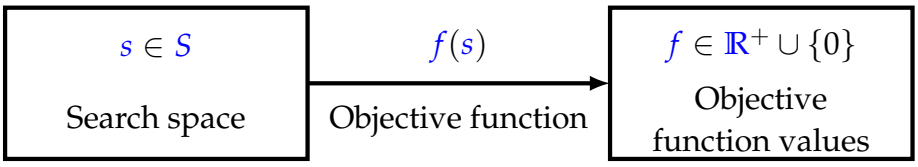


Figure 3.2: The mapping from feasible solutions to objective function values in a combinatorial optimization problem

- *(hard)* constraints $hc_1, hc_2, \ldots, hc_{nc}$, *where* $hc_i : V \mapsto \{0, 1\}, 1 \leq i \leq nc$;
- *a set of* feasible solutions $S = \{s | s \in V, \prod_{i=1}^{nc} hc_i(s) = 1\}$;
- *a real value* objective function f *to be minimized*[24], *where* $f : S \mapsto \mathbb{R}^+ \cup \{0\}$.

The main mapping in a combinatorial optimization problem is the objective function, as shown in Figure 3.2. To solve a combinatorial optimization problem one has to find a (feasible) solution $s^* \in S$ with minimum objective function value, that is, $f(s^*) \leq f(s)$ for any $s \in S$. s^* is called a *global optimum* (or globally optimal solution) of the problem; and the set $S^* \subseteq S$ consisting of all possible s^* is called the set of *global optima*.

Some combinatorial optimization problems are not difficult to solve, especially for those having an optimal substructure. For example, the problem of finding the shortest path in a finite graph can be solved by Dijkstra (1959)'s algorithm in $O(n^2)$ time. However, many other combinatorial optimization problems are in the NP-Complete class. Any NP-Complete problem, theoretically, can be transformed to any other NP-Complete problem in (deterministic) polynomial time. For example, Ruiz-Vanoye et al. (2011)

[24] As maximizing an objective function f is the same as minimizing $-f$, this thesis mainly focuses on minimization problems without loss of generality.

summarized transformations between hundreds of NP-Complete problems, where many problems were combinatorial optimization problems.

3.2.2 Hyper-heuristics

Hyper-heuristics aim at selecting or generating heuristics to solve problems. The process of selecting or generating heuristics is regarded as a machine learning procedure in this thesis. The definition of the hyper-heuristics is given as Definition 2, which is partially based on Rice (1976)'s definition of algorithm selection. A similar mapping model can be found in Ochoa, Vázquez-Rodríguez, Petrovic, and Burke (2009).

Definition 2 (Hyper-heuristics)**.** *A hyper-heuristic is a 3-tuple* $(ag, \vec{A}, l)$ *in the model of Figure 3.3, where:*

- S *is the set of* search space *of a given combinatorial optimization problem as defined in Definition 1;*
- $\vec{A} = \{\vec{a}_1, \vec{a}_2, \ldots, \vec{a}_{na}\}$ *is a set of* attribute vectors*;*
- $ag : S \mapsto \vec{A}$ *is a mapping of* attribute generation*, sometimes* ag *can return multiple attribute vectors;*
- $H = \{h_1, h_2, \ldots, h_{nh}\}$ *is a set of* low-level heuristics *(LLHs);*
- l *is a machine learning procedure that predicts the best LLH or generates a new LLH according to all vectors of attributes;*
- $\pi : H \times S \mapsto \mathbb{R}^+ \cup \{0\}$ *is a mapping of* performance measure*.*

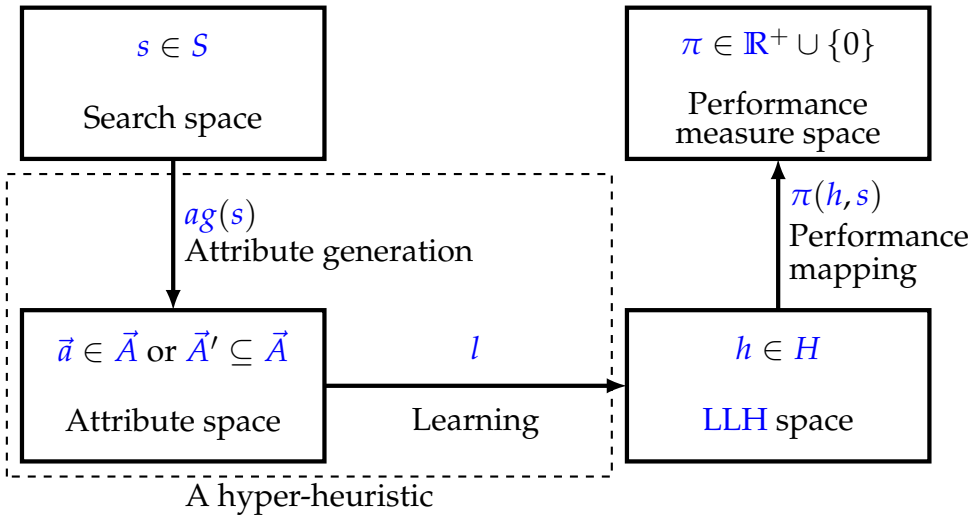


Figure 3.3: Schematic diagram of a hyper-heuristic

In Definition 2, the learning l is the core of a hyper-heuristic. However, the resulting LLHs of a hyper-heuristic do not necessarily guarantee their solutions to be optima. In other words, the best algorithms in the LLH space that maximize π are usually not guaranteed by a hyper-heuristic. The hyper-heuristics that returns better LLH h in general are preferred naturally in optimization practice. If someone considers deciding on an optimum hyper-heuristic as a new optimization problem, the objective is to minimize π with the ag, $\vec{A}$ and l being three variables to determine.

In many heuristic selection approaches, properties of LLHs such as the computation time were chosen as decision attributes. The values of objective functions were considered as the labels (or part of labels) for supervised learning. A typical result of learning may look like "if a particular condition is met, return the first LLH; otherwise return the second LLH." It should be noted that hyper-heuristics *can* optimize continuous optimization problems. However, the problems are assumed as combinatorial optimization in the

following, because of the objective domains of the SPOT method.

3.2.3 The U/EA and the U/EA^2 standards

A U/EA (Unconstrained and with Equinumerous Assignments) standard (or model) is proposed for the combinatorial optimization problem to enable the approximation in the SPOT algorithm. There are two restrictions in the U/EA standard. The first restriction is that there are no inter-variable hard constraints. In other words, changing the assignment of each variable is *unconstrained* by others. The other restriction is the equinumerous assignments, which means the domain of each variable has the exactly same cardinal number.

Definition 3 (Equinumerous assignments)**.** *Two variables x_i and x_j in Definition 1 have* equinumerous assignments, *if and only if $\|D_i\| = \|D_j\|$, $i \neq j$, $x_i, x_j \in X$.*

Definition 4 (U/EA problem)**.** *A combinatorial optimization problem in Definition 1 is called Unconstrained and with Equinumerous Assignments (U/EA), if and only if:*

1. *there are no hard constraints among different variables in the given problem; and*
2. *all variables have equinumerous assignments in the given problem.*

Definition 5 (U/EA transformation)**.** *In the model of Figure 3.4, the injective function T from the solution space of a combinatorial optimization problem is a U/EA (Unconstrained and with Equinumerous Assignments) transformation, if and only if the image of T fulfills the definition of U/EA in Definition 4.*

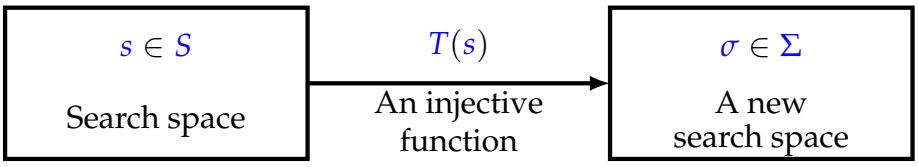


Figure 3.4: The mapping of U/EA transformation from the search space of a combinatorial optimization problem

As a mapping, a U/EA transformation can be either an equivalent (usually bijective) transformation or a relaxation (usually injective) technique. In general, relaxation is much easier to implement in practice. For example, one naive and effective relaxation technique is directly removing all the constraints among the variables.

An example of a U/EA problem can be given in the TSP domain. The solution (with n edges) to a TSP is required to be a Hamiltonian cycle (connected graph) as a hard constraint. A possible relaxation is "to find n edges to form a graph so that each city has a degree of 2". In the relaxed model, the graph is still with n vertices and n edges, but the connectivity of the graph is not necessarily held.

Another standard (or model) called U/EA^2 is presented to regulate the attributes in the SPOT algorithm further. In the U/EA^2 standard, each attribute vector $\vec{a} \in \vec{A}$ has the same number of columns (dimensions), and all the attributes in one column are consistent and have the same units.

Definition 6 (Equinumerous attribute vectors)**.** *Two attribute vectors $\vec{a}_i$ and $\vec{a}_j$ are called* equinumerous *if and only if they have* equal *dimensions (amount of columns) over a certain field, i.e.* $\dim_{\mathbb{R}} \vec{a}_i = \dim_{\mathbb{R}} \vec{a}_j$.

Definition 7 (U/EA^2)**.** *An attribute space in Definition 2 is called Unconstrained and with Equinumerous Assignments and Equinumerous Attributes (U/EA^2), if*

and only if:

1. *all the attribute vectors are generated from a* U/EA *combinatorial optimization problem; and*

2. *all the attribute vectors are equinumerous attributes.*

3.2.4 The SPOT hyper-heuristic

Definition 8 (SPOT hyper-heuristic)**.** *The SPOT hyper-heuristic is a 7-tuple* $(sam, S', T, \Sigma, ag, \vec{A}, l)$ *as shown in the model of Figure 3.5, where:*

- $sam : S \mapsto S'$ *is a mapping of* sampling *which returns a small-scale* ($n' \ll n$) *subproblem;*

- S' *is the solution space of the subproblem;*

- $T : S' \mapsto \Sigma$ *is a mapping of* U/EA *transformation;*

- Σ *is the solution space of the transformed problem,* $\|\Sigma\| \ll \|S\|$*;*

- $\vec{A} = \{\vec{a}_1, \vec{a}_2, \ldots, \vec{a}_{na}\}$ *is a set of attribute vectors, where each vector stands for one specified situation for one possible assignment competing with all other competitors, and* $\vec{A}$ *covers all the possible situations,* $na \geq n \gg n'$*; and*

- $ag : \Sigma' \mapsto \vec{A}^{\sum^{n'} D_i}$ *is a mapping of assignment-based attribute generation which generates a set of* $\sum^{n'} D_i$ *(as many as the size of all possible assignments in the subproblem) attribute vectors from one solution, the attributes in each vector are fully determined by the situation in regard to the competition of an assignment.*

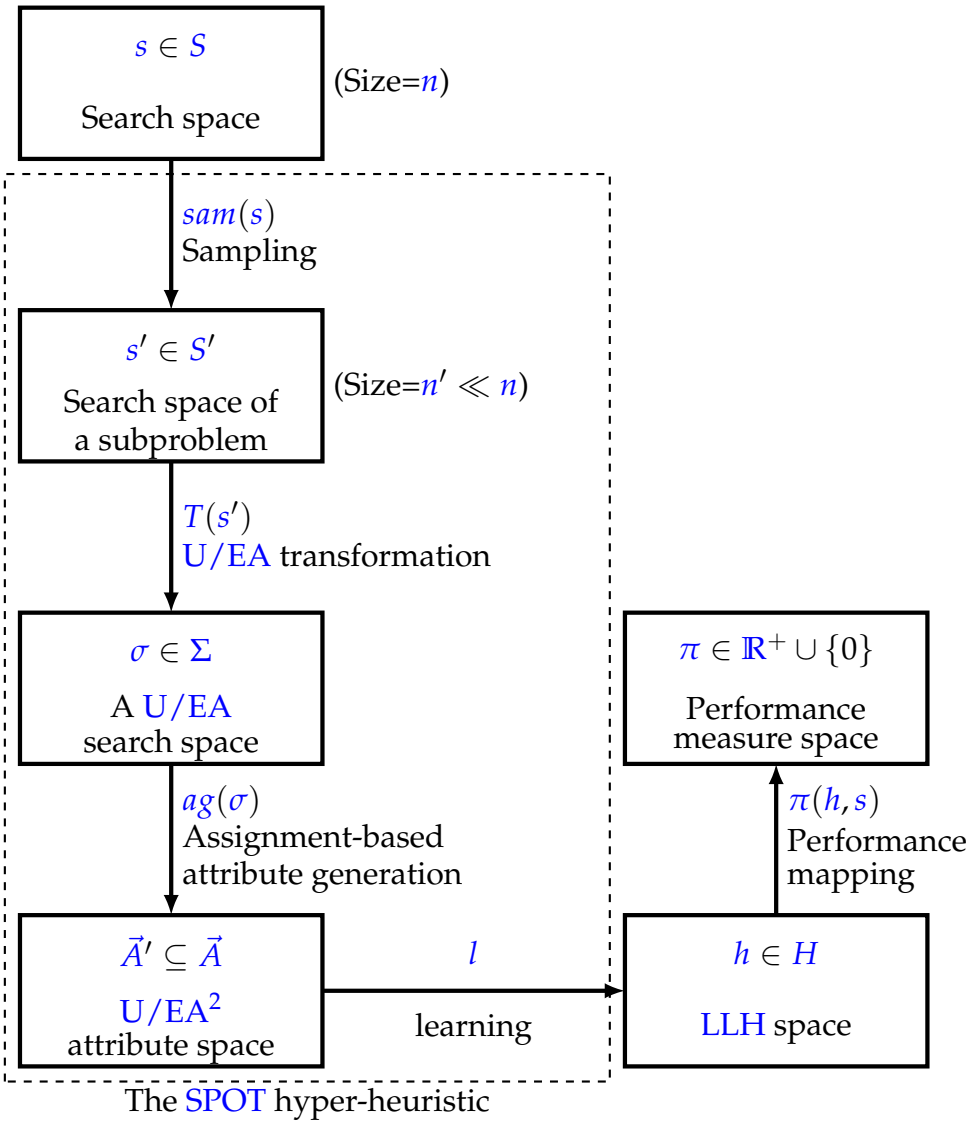


Figure 3.5: Schematic diagram of SPOT hyper-heuristics

The other notions are the same as those in Definition 2.

Theorem 3. *The SPOT algorithm is a special case of hyper-heuristics.*

Proof. Define a mapping $ag'(s) = ag\left(T\left(sam(s)\right)\right)$, then the SPOT hyper-heuristic $\langle sam, S', T, \Sigma, ag, \vec{A}, l\rangle$ is a hyper-heuristic $\langle ag', \vec{A}, l\rangle$. □

In the SPOT hyper-heuristic, each possible attribute vector $\vec{a} \in \vec{A}$ stands for a particular possible situation of an assignment competing with all other competing assignments. The returned algorithm h can generally assign each attribute vector $\vec{a}$ with a prediction of a (computable, usually finite) real-valued value $\gamma \in \Gamma = [0, 1]$, where Γ is the set of possible predictions. An example is the "likelihood of appearing in suboptima". Therefore, the general cardinal number of H is $\|H\| = \|\Gamma\|^{na}$. In other words, there could be as many as $\|\Gamma\|^{na}$ possible LLHs in total. In this thesis, $na \geq n$ is assumed to be true. See Section 3.4 for more details on this assumption.

3.2.5 Heuristic selection and heuristic generation

Distinguishing heuristic selection and heuristic generation is actually not easy due to various transformations (and equivalences) between different problems. For example, the parameter optimization methods mentioned in Section 1.3 can be regarded as complicated algorithm selection procedures. An example of parameter optimization is "to determine two 0-1 integer parameters for an algorithm". There are four possible configurations, including (0, 0), (0, 1), (1, 0) and (1, 1). The problem can be seen as selecting a single best algorithm from four. An algorithm generation approach, on the other hand, seems difficult in becoming an equivalent selection procedure. An example of algorithm generation is "to find the best propositional formula on n (Boolean) propositional variables for fitting a Boolean function". There are 2^n irreducible formulae standing for 2^n different functions. If every function is expected to be tried in a test-and-select fashion, the problem can be seen as selecting the best algorithm from 2^n algorithms.

A meaningful boundary between selecting an algorithm from four and selecting one from 2^n is the *computability* of computers. Because the size n of a combinatorial optimization problem is countable, as defined in Definition 1, i.e., $n \leq \aleph_0$. With the growth of n, the LLH space of a generation method grows explosively and becomes incomputable so that it cannot be enumerated and tested singly on a Turing machine. "Generation" or "make-up" behavior, instead of the test-and-select method, is necessary for the feasibility of returning heuristics in practice, and is called "generation". Otherwise, an exact (full) exploration of the LLH space must cost explosive time[25].

Definition 9 (Heuristic selection)**.** *A hyper-heuristic in Definition 2 is a* heuristic selection *algorithm in a domain of combinatorial optimization problems, if and only if* $\|H\| \leq \aleph_0$ *can be held for* all *the problems in that domain, necessarily including those with* $n = \aleph_0$*, where* H *is the LLH space of the hyper-heuristic.*

Definition 10 (Heuristic generation)**.** *A hyper-heuristic in Definition 2 is a* heuristic generation *algorithm in a domain of combinatorial optimization problems, if and only if* $\|H\| > \aleph_0$ *can be held for no less than one problem with* $n = \aleph_0$ *in that domain, where* H *is the LLH space of the hyper-heuristic.*

Typical examples of heuristic selection algorithms and heuristic generation algorithms can be found in Section 2.2.1 and Section 2.2.2, respectively. The definitions given in this chapter clearly show the intensions and outline the extensions of heuristic selection and heuristic generation. The definitions are compatible with previous research, such as Burke, Hyde, Kendall, Ochoa, Özcan, and Woodward (2010). In fact, every heuristic selection algorithm

[25]This could probably be a reason why there was no exact algorithm generation method in literature, as far as concerned. See Figure 1.7.

can be equivalently transformed to the optimization of a parameter tuning problem with finite independent parameters according to the definitions, while every heuristic generation algorithm can be equivalently transformed to the optimization of a parameter tuning problem with infinite independent parameters. See Section 3.4 for more details.

Definition 11 (Turing (1936)'s computable number)**.** *A number is* computable *if it differs by an integer from the number computed by a circle-free machine.*

Lemma 1. *The cardinal number of the set* $\mathbb{C}$ *of* all computable numbers *is* $\aleph_0$*.*

Proof. There are countably many Turing machines. Thus $\mathbb{C}$ is countably infinite at most ($\|\mathbb{C}\| \leq \aleph_0$). The set of all integers is a subset of the set of all computable numbers ($\mathbb{Z} \subseteq \mathbb{C}$). Thus $\|\mathbb{C}\| \geq \|\mathbb{Z}\| = \aleph_0$. Therefore, $\|\mathbb{C}\| = \aleph_0$. □

For example, all rational numbers, $\sqrt{2}$, $\log_5 8$, π and e are computable numbers. However, in fact, most irrational numbers are incomputable. It should be noted that incomputable irrational numbers are rarely involved in the applications and research of combinatorial optimization. Therefore, algorithms involving incomputable numbers are not discussed in the following. Turing (1936), Davis (1965), Ko and Friedman (1982), Soare (1996) provide more information about computability and computable numbers.

Theorem 4. *An algorithm selection method for heuristics with a finite (or countably infinite) set of ready-to-use heuristics is a* heuristic selection *algorithm.*

Proof. The LLH set H is the same as the set of given ready-to-use heuristics. Therefore, $\|H\| \leq \aleph_0$. □

Theorem 5. *A parameter optimization method for heuristics with a finite number of computable real parameters is a* heuristic selection *algorithm.*

Proof. Let a be the cardinal number of all parameters, $\|H\| \leq \|\mathbb{C}\|^a \leq (\aleph_0)^a = \aleph_0$. □

Theorem 6. *The SPOT hyper-heuristic is a* heuristic generation *algorithm.*

Proof. The SPOT hyper-heuristic is a hyper-heuristic (Theorem 3). It is clear that $na \geq n$ is assumed to be true[26]. For the set of possible predictions for each attribute vector, $\|\Gamma\| \geq 2$ can be generally held, where Γ is the set of possible values for each attribute vector in the learning result. The cardinal number of H is $\|H\| = \|\Gamma\|^{na} \geq 2^{na} \geq 2^n = 2^{\aleph_0} > \aleph_0$, for all $n = \aleph_0$. □

The incomputability (or undecidability) involved in heuristic generation algorithms, in fact, has two sides. On one hand, the incomputability of the (very large-scale) LLH space makes it possible to return *new* results that have not been unexamined by previous researchers. On the other hand, the incomputability brings difficulties (mainly on complexity) in the practice of optimization. The incomputability can, sometimes, be constrained manually so that a "promising" part of the LLH space can be focused. One example of restrictions to the "squeaky wheel" optimization is "to determine the top five 'bottleneck' spots" (in $\Theta(n^5)$ class, where n is the number of variables) instead of "determining the full priority sequence" (in $\Theta(n!)$ class) in each iteration. Clearly, the new constrained "squeaky wheel" optimization focuses on a "promising" part and becomes a heuristic selection according to the definitions.

[26] See Section 3.2.4 and Section 3.4.

3.3 Methodology of Developing a SPOT Hyper-Heuristic

In general, there are two major phases in developing the SPOT hyper-heuristic: the *design* phase and the *run* phase. In the first phase, the input problem is remodeled to meet the standards of U/EA and U/EA^2. Domain-specific parameters, including numeric and non-numeric parameters, are determined and fine-tuned. The plans for applying the results of supervised learning are also designed in the first phase. The latter phase is much more straightforward. Given an input problem, the SPOT hyper-heuristic can generate a number of new LLHs for solving it. The instance information is learned in the latter phase.

3.3.1 The design phase

There are three phases in developing a SPOT hyper-heuristic, as shown in Table 3.1. A seemingly unusual thing is that the development should stop in certain cases after the first phase. That means the SPOT algorithm cannot handle the input problems well, because of three possible reasons:

- The input problems do not meet the requirements of the U/EA standard;
- The attributes for supervised machines learning do not meet the requirements of the U/EA^2 standard; and/or
- The correlations among variables are too high so that the suboptimum- and proportion-based learning always fails to approximate the correct classifiers to identify (sub)optimum assignments.

Table 3.1: Main phases in the design phase in developing a SPOT hyper-heuristic

Phase	Brief content	Successful results	To do when fails
P1	To try to transform the given problem	A U/EA problem & a U/EA^2 attribute space	To quit for other methods
P2	To determine parameters	A set of parameters	To try different plans
P3	To modify or to create LLHs	New LLHs	To try different plans

3.3.1.1 P1: Transformations

In the first phase (P1), the main concern is on transformation. One transformation aims at converting an input combinatorial optimization problem to a U/EA problem. This step can be omitted if the given problem has occasionally met the two requirements of U/EA. The other transformation aims at determining decision attributes and generating one attribute vector for each possible assignment. In certain cases, the two transformations cannot be conducted, which means the SPOT algorithm would not be able to handle the problem well.

Another requirement besides the two standards is the low-correlation assumption. An indicator $r : S \times S \mapsto [0, 1]$ of "resemblance" is introduced to measure the percentage of same assignments of two different solutions.

$$r(s, s_0) = \frac{\text{Number of same assignments}}{n}. \tag{3.1}$$

According to the assumption, the r value between any two suboptima of a given instance should be close to 1. Furthermore, the average resemblance of every (different) problem instance in the same domain should be very close to 1, in the ideal cases.

Definition 12 (Compatibility of U/EA problems)**.** *A domain of problems which meet the U/EA standard are called* "compatible" *with the SPOT hyper-heuristic, if and only if for given a set* $\{\bar{r}_1, \bar{r}_2, \ldots\}$ *of average resemblances of suboptima of different instances (one average resemblance for each instance) and the overall average resemblance* $\bar{r}$ *for all instances, the following conditions can generally be held:*

- *(*Generality*)* $\bar{r}_i \approx \bar{r}_j \approx \bar{r}$*, where* $\bar{r}_i, \bar{r}_j \in \{\bar{r}_1, \bar{r}_2, \ldots\}$,
- *(*Approximability*)* $\bar{r} \approx 1$.

A high $\bar{r}$, such as 0.8, denotes the assignments in suboptima are hopefully well approximable through investigating the subproblems and suboptima. A low average resemblance (e.g., 0.15) means it is probably not reliable to approximate in this way. A high level of generality means that the difference between learning from a problem instance and learning from its subproblems is small. A high level of approximability denotes the difference between learning from optima and learning from suboptima is small. The compatibility of the U/EA standard enables[27] feasible and reliable suboptimum- and proportion-based learning. In many perturbative meta-heuristics, such as the Lin-Kernighan local search for TSP mentioned in Section 2.1.1, new suboptima are expected to be reached from a known suboptimum by modifying a few assignments. A high level of compatibility is usually closely related to a domain with successful perturbative methods. Generally, either a low approximability or a low level of generality, or both, denote that the

[27] In fact, the compatibility could be the assumption of the proposed methodology. However, it is introduced as a "soft" indicator other than a "hard" assumption for the purpose of a friendly cross-domain foundation.

input problem is barely compatible with the assumptions of the SPOT hyper-heuristic. The developer should try another transformation or change to another algorithm.

The other standard to be determined in P1 is the U/EA^2 model. A set of attributes needs to be designed to represent the competition features of the assignments, so that the sampled subproblem and its suboptimum can be transformed into training data. A label column is determined by the training suboptimum. The trained classifiers can predict the unlabeled assignment in the given problem. The principle of the design of this model is to include enough information for supervised learning. The state-of-the-art supervised learning can precisely remove the useless and redundant decision attributes in the results. It is usually not necessary to validate attributes design carefully with indicators.

3.3.1.2 P2: Parameter determination

The second phase (P2) aims at determining numeric and non-numeric parameters. There is one important numeric parameter and one important non-numeric parameter to determine and to fine-tune the approach. Both are sampling parameters. The numeric parameter is the size n' of the subproblem to be sampled. The non-numeric parameter is the method of sampling. With the two parameters, a subproblem can be generated. Other parameters are "secondary" parameters, for example,

- an effective problem algorithm[28] which can find one or more suboptima

[28] Determination of an algorithm can also be regarded as a determination of a non-numeric parameter.

for the subproblem and its parameters, and

- a supervised machine learning method and related parameters to discover instance-specific knowledge.

Above parameters are called "secondary", because they usually have been well studied. The effective methods and suggested values of parameters are well-known in many domains. In development, the secondary parameters should be determined before the two important parameters of sampling. Some secondary parameters, such as the problem solving algorithm, can be strongly domain-dependent. The design and fine-tuning of the domain-dependent secondary parameters can be referred to the domain literature. The machine learning is relatively independent of the problem domain, if the training data (including decision attributes) is conducted for a general-purpose machine learning. In this thesis, three machine learning methods, including C4.5, Naïve Bayes and Ripper, are employed together to discover knowledge simultaneously. The three methods are very efficient according to literature and come from the three main branches of supervised learning techniques, as shown in Figure 2.1. The final result of classification is the average value of the three methods for each unlabeled row (assignment). It is believed that the average result of the three is more robust. The parameters for the three methods are borrowed from suggested values by Weka, as shown in Section 4.3.1. The result of machine learning can predict "what assignment ought to be in optima or suboptima".

In P2, the parameter n' is determined and fine-tuned first. The way method sampling is set to "random" selection, having an average effective n' in general. A verification of this development can be found in Section 6.5.

With determined standards and secondary parameters (inclining subproblem solving and learning), one most favored assignment can be identified from a group of competitors for each variable. The resemblance r is used to measure the resemblance between the identified most favorite assignments and a known suboptimum. A higher level of r denotes, hopefully, more improvements by the final generated LLHs. Different values of n' can be tested for selecting one value that maximizes r. Generally, when n' increases, r increases and the time cost of the subproblem solving and the learning increases (dramatically), too. Therefore, the developer should balance the size n' to make sure r is high enough, and the time cost is limited. In practice, if n' is set to a constant value, the overall time cost of the SPOT heuristic generation will probably be in polynomial-time.

Random selection should generally be a feasible and effective sampling method. The reason lies in the correlation-free condition among the variables in the U/EA standard. In random selection, a proportion of the given problem is made in terms of the designed group randomly[29]. Furthermore, certain sampling methods can also be investigated to fine-tune the sampling and the classifiers, especially when some inter-variable hard constraints have been ignored in P1. Examples are a cropped area of cities in an Euclidean TSP, and a cropped (nearest) segment of job permutations. Different sampling methods can be tested. One or more methods can be selected to maximize the indicators r in general. It should be noted that some groups of competitor assignments would partially change after the sampling.

[29]The random choices were based on the pseudo random number generator in computer programs. Therefore, the generated numbers were not completely random in theory.

3.3.1.3 P3: Heuristic modification

The third phase (P3) is the "last mile" and aims at transforming the instance-specific results of learning (classifiers) back to LLHs. There basically are two main ways of generating LLHs: *modifying* and *creating*. *Modifying* LLH means that the generated LLH is fully or partially based on existing heuristics. If taking a heuristic as a search sequence that traverses a limited scope of the search space, one can principally modify the heuristic in two ways:

- to change the search *scope* and
- to change the search *sequence*.

An example for the first is improving the 2-Opt local search to the 3-Opt local search in the TSP domain. An example of the latter is reordering the Lin-Kernighan local search for the TSP, which originally prefers to try high-priority (short edges) swaps first. A modified version prefers to try the most favored edge determined by learning results first. *Creating* LLH denotes that the generated LLH is fully constructed by the classifiers. A naive (but sometimes ineffective or infeasible) LLH is: to select a variable randomly, to choose the most favorite assignment from the group regarding the hard constraints, and to repeat until a solution is built. The indicators in P3 are the traditional solution quality and the time cost. According to experience, modifying methods are more efficient than creating in general, therefore, the thesis focuses on modifying LLHs.

In conclusion, the first and the last phases are about transformation. The two phases are relatively difficult to handle. There are only two main parameters to be determined, one after the other, in the second phase. The expected final performance can be measured by a numeric indicator. Therefore, the second phase is relatively easier. Some important components, such as the standards and subproblem solving algorithm, are domain-specific. In other words, the domain-specific parameters must be redesigned when the SPOT hyper-heuristic is applied to a new domain.

3.3.2 The run phase

It is more straightforward to solve a problem than to design and fine-tune the SPOT algorithm. The procedure of solving can be briefly described as follows. Given a problem P, There are four phases for generating new LLHs:

- To start with a problem P and its unlabeled attributes about assignments,
- To sample for a subproblem P' with determined size n' and to solve the subproblem efficiently to a suboptimum with a determined algorithm,
- To generate labeled attributes referring to the suboptimum,
- To learn instance-specific knowledge (as classifiers) on what assignments are likely to appear in suboptima with three methods, C4.5, Naïve Bayes and Ripper, to predict the labels of unlabeled attributes in phase 1., and to modify certain heuristics for new LLHs according to the labels.

It should be noted that the last step consists of a process of finding instance-specific knowledge. Therefore, the generated LLHs become instance-specific algorithms and might not be useful in solving other problem instances.

3.4 Discussion

There are some similarities between the proposed SPOT algorithm and the methodology in Xue, Chan, et al. (2011). For example, they have similar concepts in the sampling subproblem and suboptimality, parameters, indicators and heuristic generation processes. Both generate instance-specific algorithms. The main difference is that Xue, Chan, et al. (2011) presented a *heuristic selection* approach, though the instance-specific information was obtained by a class association rules learning. There were fourteen prima-facie decision attributes and one label defined. Each attribute has a finite set of values. The classifier was in the form of class association rules with confidences. The number of possible classifiers was countable. Thus, the number of possible LLHs was countable. The method in Xue, Chan, et al. (2011) was also specialized in regard to the Euclidean TSPs. In comparison, the SPOT algorithm employs a semi-automatic attribute generation (See Section 4.3.1) and three different supervised learning methods for cross-domain development. The learning module in the SPOT hyper-heuristic is expected to be more open and robust.

The idea of dividing (sampling) and analyzing a proportion for solving a given problem is not novel. It can be found in many heuristic and exact algorithms, such as the first-come, first-served. It can also be found in the

early days of differential calculus, ancient art of war and activities even earlier. It is a natural thinking in problem solving. In this thesis, a standard U/EA is given to regulate the input for the sampling proportion. The standard and another property, compatibility, guarantee high approximability and low correlations of variables for sampling.

If a given domain holds a good approximability to the U/EA standard, heuristic (or at least perturbative) algorithms and their decision attributes should not be too difficult to find in literature. The initial raw attribute can be set to a possibly maximal set. The attribute selection procedure in SPOT can help in choosing the best raw attributes for your domain.

It should be noted that the number na of all possible attribute vectors is determined by the definition of the attributes. Generally, there are three cases when $n = \aleph_0$, listed as follows:

- If there is at least one attribute having countably infinite possible values, such as an integer or a real number, then na must be countably infinite.
- If there are a countably infinite number of attributes, and each attribute has a finite definition domain, then na must be countably infinite.
- If the number of attributes is finite and the definition domain of each attribute is also finite, such as three Boolean attributes, then na must be finite.

The inequality $na \geq n$ *may not* be held in the last case. In fact, a natural attribute for an assignment in an NP-Complete problem usually has no less than n or infinite possible values, such as the distance in TSP and the

processing time in FSP. In addition, attributes are defined to distinguish the scenario of the problem so that some classifiers can be built. Therefore, a design in the last case probably cannot meet the demand of resolution and distinguishability for NP-Complete problems. Therefore, $na \geq n$ is assumed to be true in this thesis.

The definitions of heuristic selection and heuristic generation are consistent with the existing taxonomy in the literature, such as Burke, Hyde, Kendall, Ochoa, Özcan, and Woodward (2010) in most cases. In fact, the extensions of both concepts are extended. For example, the parameter tuning techniques which used to be out of hyper-heuristics are regarded as heuristic selection in this thesis. It should be noted that the two concepts are only distinguishable on problem domains that can be extremely large ($n = \aleph_0$). That means, if a problem domain contains a strictly finite (or limited number of) variables, any hyper-heuristic on the domain cannot be considered as either heuristic selection or heuristic generation in this study. However, this situation is rare in typical combinatorial optimization problems. Therefore, the issue is not addressed in detail, and further research could be carried out in the future. A possible plan is to define the hyper-heuristic approaches on an always finite problem domain with finite variables as heuristic selection, according to computability.

Chapter 4

Java Implementation of the SPOT Hyper-Heuristic

Die Philosophie ist keine Lehre, sondern eine Tätigkeit.

Philosophy is not a body of doctrine but an activity.

Tractatus Logico-Philosophicus

Ludwig J. J. Wittgenstein

The design and implementation of the proposed SPOT algorithm are presented in this chapter. Two algorithm development libraries, HyFlex and Weka, are briefly introduced in Section 4.1. Section 4.2 represents the design classes and public functions in the SPOT algorithm. The implementation of two cross-domain classes and their functions, such as the training of classifiers and the general procedure of problem solving, are detailed in Section 4.3. Section 4.4 presents some overall discussion.

4.1 Supporting Libraries

This section introduces two algorithm development libraries: HyFlex and Weka. HyFlex is a cross-domain hyper-heuristic development library and Weka is a machine learning software and machine learning development library. Both are written in Java language and both support cross-platform development.

4.1.1 HyFlex

HyFlex[30] (Hyper-heuristics Flexible framework) is a Java cross-domain hyper-heuristic development platform (Ochoa, Hyde, et al., 2012). HyFlex was also the competition platform in the hyper-heuristic competition CHeSC 2011 which focused on heuristic selection approaches. This thesis adopts the same version of HyFlex as in CHeSC 2011. There are two main parts designed in HyFlex for general heuristic selection approaches (Ochoa, Hyde, et al., 2012):

- *cross-domain* part of hyper-heuristic algorithms, and
- *domain-specific* part of LLHs.

There is a "domain barrier" assumed between the two parts. The cross-domain part, which is capable of calling ready-to-use LLHs through normalized public functions, is considered the active part. The domain-specific part is considered the passive part. Particularly, there are two optional parame-

[30] See http://www.asap.cs.nott.ac.uk/external/chesc2011/hyflex_description.html.

ters, depth of search and intensity of diversification, for each LLH. An LLH may use none, one or two of the parameters.

Two super classes are defined as interfaces for the hyper-heuristics. One is the HyperHeuristic which aims at encapsulating the data and operations in the cross-domain part. Another is the ProblemDomain which emphasizes on including the *domain-specific* LLHs and their data structures. The data structures are completely inaccessible for the HyperHeuristic class. A hyper-heuristic approach implemented in HyFlex can interact with the domain-specific LLHs through public function calls. Public interface functions in HyperHeuristic are mainly designed for supporting high-level strategies, such as setting a time limit, tracing changes of objective values and getting the best-so-far objective values. Public interface functions in the ProblemDomain include getting and setting the parameters, loading problem instances, listing and calling available LLHs and initializing and copying solutions in the memory. More details about HyFlex can be found in Ochoa, Hyde, et al. (2012).

4.1.2 Weka

Weka[31] (Waikato Environment for Knowledge Analysis) is a machine learning software and a development library in Java (Witten & Frank, 2005). The version used in this thesis is 3.6. Weka contains many well implemented algorithms for data analysis and machine learning, such as most of the supervised learning methods shown in Figure 2.1. Weka also provides visualization tools and graphical user interfaces to make the algorithms easy

[31]See http://www.cs.waikato.ac.nz/ml/weka/.

to use. The algorithms in Weka accept training data and test data as tables of values, where each column stands for a decision attribute (or a label) and each row denotes a training (or test) record. This study employs three learning methods, J48, NaiveBayes and JRip, from Weka. The three classifiers are Java implementations of the well-known C4.5, Naïve Bayes and Ripper methods introduced in Section 2.3.1. The parameters of the three methods are the default values given by Weka. A heuristic *attribute selection* method BestFirst is also used to choose a few of the most informative attributes when the amount attributes are too large.

4.2 The Class Design

The operations in the SPOT algorithm are designed and encapsulated in three classes, ProblemDomain_UEA2, SPOT_ML and SPOT, as shown as n the thick borders in Figure 4.1. The three classes are introduced in the following.

The class ProblemDomain_UEA2 implements the super class ProblemDomain in HyFlex for supporting the domain-specific operations in the SPOT hyper-heuristic, such as the sampling for subproblems (`genSubprob`()), the U/EA transformation, the generations of raw attributes[32] and the modifications (or creations) of LLHs. The operations on transformation and attributes are encapsulated in public functions, such as manipulating competitive groups (`getCompetitiveGroups`()), raw attributes (`getRawAttributes`()) and labels (`getIntLabels`() and `setIntLabels`()).

[32]The first step of attribute generation, see Section 4.3 for more details.

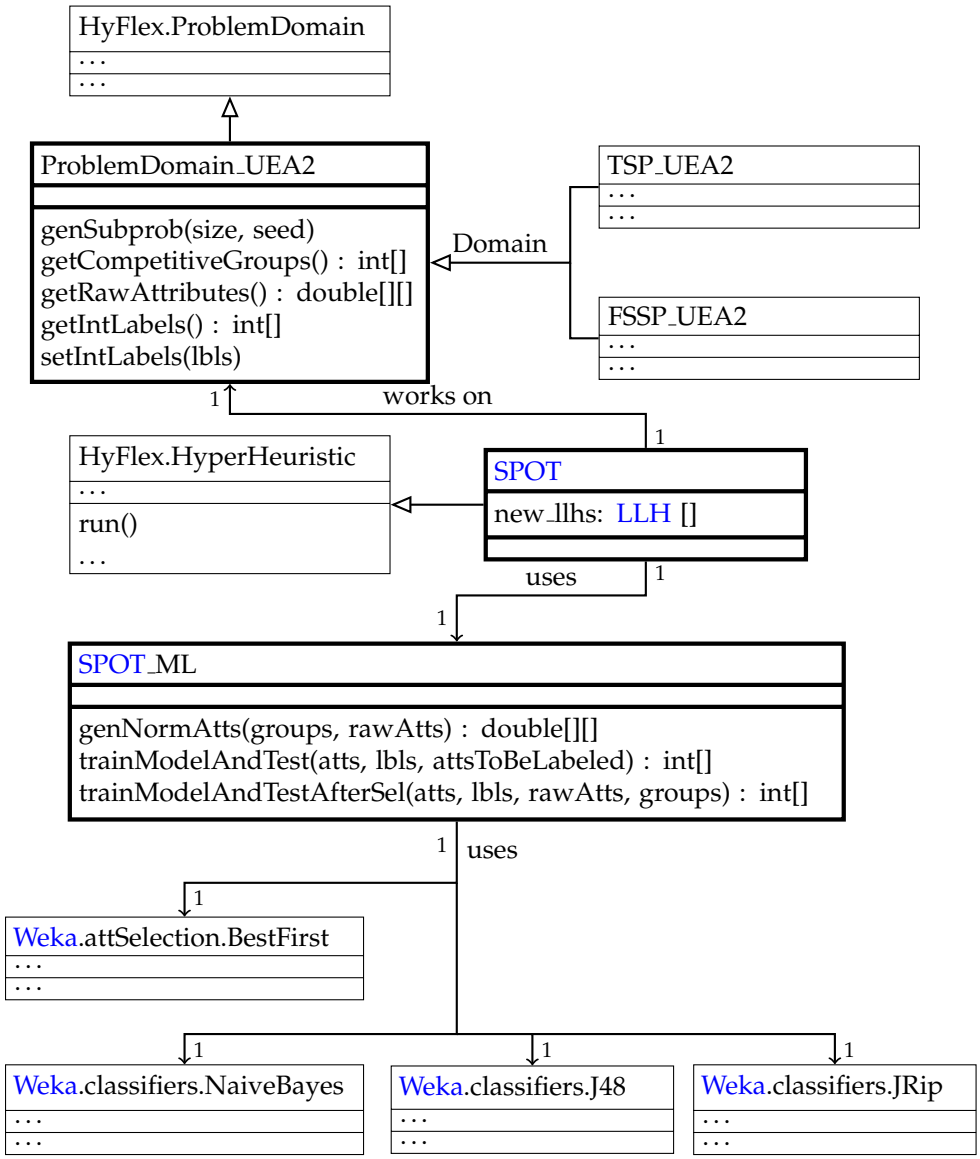


Figure 4.1: The class diagram of the SPOT hyper-heuristic

The class SPOT_ML includes the cross-domain operations about attribute generation and training and applying classifiers. `genNormAtts`() is a function that populates a number of attributes by comparing a raw attribute of an assignment to those of its competitors, see Section 4.3. SPOT_ML encapsulates three effective classification techniques, i.e. J48, NaiveBayes and JRip, in the function `trainModelAndTest`(). The attribute selection

method BestFirst is encapsulated into `trainModelAndTestAfterSel()`, which extends the function `trainModelAndTest()`.

The class SPOT is the high-level controller. It inherits the standard functions from the super class HyperHeuristic. The class SPOT accepts an instance of ProblemDomain_UEA2 as its input problem and solves it step by step, see Section 4.3 for the details of the steps. The learning in the class SPOT is powered by the functions of SPOT_ML.

4.3 Implementation in Java

The three classes were implemented in Java 2 Platform Standard Edition (SE) version 1.6.0_32 (64-bit). The compiler was the default compiler provided in Java 2 SE. The machine was an Intel Core i7 975 3.33 GHz[33]. The operating system was a Windows 7 64-bit with Service Pack 1. The class ProblemDomain_UEA2 is relatively domain-specific. The examples of its implementation can be found in Section 5.2 and Section 6.2. The classes SPOT_ML and SPOT are cross-domain. Their implementations are detailed as follows.

4.3.1 The class SPOT_ML

There are three main public functions to implement in SPOT_ML. Define a problem with n groups, p competitor assignments in each group and ra different raw attributes (measures) for each assignment. The function

[33]The official test program showed that 1.00 CPU second in four parallel thread mode for this CPU was equivalent to 1.00 CPU second for a Pentium 4 3.0 GHz.

`genNormAtts`() generates a table of normalized attributes with $ra \times (2p+3)$ columns and $n \times p$ rows. Each row stands for the competition between one assignment and its competitors. To illustrate the generation of the columns, it is assumed that there is only $n = 1$ group of p assignments and $ra = 1$ raw attribute (measure). The raw attributes of the assignments are measured and put in an ordered set $\{x_1, x_2, \ldots, x_p\}$, where $x_1 \leq x_2 \leq \ldots \leq x_p$. For each i-th assignment, Table 4.1 shows $2p+3$ normalized attributes (columns) which describe the competition. A function $bound100$ is used to limit the ranges. The first generated attribute is the real value of the raw attribute. The remaining attributes are counts and comparisons without any units. A column of Boolean labels can be attached to the table, where `true` denotes the assignment can be found in a suboptima (or optima) and `false` otherwise. Particularly, when there are two assignments in each group, some columns can be removed due to redundancy. For example, the second row of attribute "repeat counter" in Table 4.1 can be omitted.

The function `trainModelAndTest`() consists of training of the classifiers and classifying of unlabeled data. It is a part of the last step in Section 3.3.2. First, the table consisting of normalized attributes and labels is used to train the three classifiers. Each training is carried out with the default parameters suggested by Weka, as listed in Table 4.2. The three trained classifiers predict the unlabeled data which is generated from the given problem by measuring the same raw attributes and generating normalized attributes. The average (or summation) value of the labels predicted by the three classifiers is regarded as the final label (prediction). In practice, non-numeric Boolean values, such as `true` and `false`, can be converted to

Table 4.1: A list of generated attributes of the i-th assignment from one raw attribute

Attribute(s)	Equation	Count
Absolute value	x_i	1
Repeat counter	$\sum_{j=1}^{ra} same^{\dagger}(x_i, x_j)$	1
Equal to the first?	$same(x_i, \text{global}^{\ddagger} \text{ minimum of } x_1)$	1
Equal to the last?	$same(x_i, \text{global}^{\ddagger} \text{ maximum of } x_p)$	1
Relative values	$bound100^{\natural}(x_i / x_j^{\sharp}), 1 \le j \le p$	p
Relative increments	$bound100(x_j / x_{j+1}^{\sharp}), 1 \le j \le p-1$	$p-1$

$\dagger$: $same(x,y) = \begin{cases} 1 & \text{if } x = y, \\ 0 & \text{if } x \neq y. \end{cases}$

$\ddagger$: The minimum or maximum is available if there are multiple groups.

$\natural$: $bound100(x) = \begin{cases} 100 & \text{if } x \ge 100, \\ x & \text{if } -100 < x < 100, \\ -100 & \text{if } x \le 100. \end{cases}$

$\sharp$: When $y = 0$, $x/y = \begin{cases} 100 & \text{if } x < 0, \\ 1 & \text{if } x = 0, \\ -100 & \text{if } x < 0. \end{cases}$

numeric values, such as 1 and 0, in order to do arithmetic operations.

A preprocessing of *attribute selection* generally can select the most informative columns to reduce the computation cost of training and to enhance the generalization of the classifiers. The attribute selection is very useful when the number of columns $ra \times (2p + 3)$ is too large (e.g., 2000). The function `trainModelAndTestAfterSel`() in the class SPOT extends `trainModelAndTest`() with an attribute selection preprocess. The preprocess calls the BestFirst search and the CfsSubsetEval evaluator provided by Weka. Details about the selection method and the correlation-based evaluator can be found in Hall (1999, Chap. 4).

Table 4.2: The default values of main parameters of the classification and attribute selection methods in Weka

Method	Main parameter	Default value	Reference
J48	Confidence	0.25	Quinlan (1993)
	Min number of objects	2	
	Number of folds	3	
JRip	Number of folds	3	Cohen (1995)
	Min number of objects	2	
	Optimizations	2	
	Seed	1	
NaiveBayes	Using kenel estimator	`false`	John and Langley (1995)
BestFirst	Lookup cache size	1	Hall (1999)
	Search termination	5	

4.3.2 The class SPOT

The class SPOT mainly implements one public interface `run`() inherited from HyFlex.HyperHeuristic, as mentioned in Section 3.3.2. The function `run`() consists of a full execution of the problem solving of the presented algorithm. Given a set of determined parameters and a problem to solve, the function `run`() can be briefly described as pseudo codes in Figure 4.2.

First, unlabeled attributes of possible assignments for classification are generated in Lines 1 and 2 in Figure 4.2. Secondly, a subproblem is sampled and is solved by a determined algorithm, as shown in lines 3 and 4. The PHunter (Pearl Hunter), which is an effective heuristic selection algorithm developed by Chan et al. (2012) on HyFlex, is employed as the input algorithm. The execution time of PHunter can be usually limited to a very short period, for example, a few seconds. Suboptima can be found in most cases, because the size of the subproblem is also limited. Then, the attributes and labels about the suboptima of the subproblem can be generated, as shown as lines

```
1. ProblemDomain_UEA2 i1 := load_problem();
2. atts1 := SPOT_ML.genNormAtts(i1.getCompetitiveGroups(), i1.getRawAttributes());
3. ProblemDomain_UEA2 i2 := i1.genSubprob();
4. i2.solved_by_algorithm(an_input_algorithm);
5. atts2 := SPOT_ML.genNormAtts(i2.getCompetitiveGroups(), i2.getRawAttributes());
6. lbls2 := i2.getIntLabels();
7. predLbls := SPOT_ML.trainModelAndTest(atts2, lbls2, atts1);
8. i1.setIntLabels(predLbls);
```

Figure 4.2: The pseudo codes of function SPOT.run()

5 and 6. Finally, the assignments of the given problem can be predicted, and some LLHs can be modified, as shown in lines 7 and 8. In the 7th line, there is also an alternative learning function `trainModelAndTestAfterSel`() consisting of a preprocess of attribute selection, as described in Section 4.3.1.

4.4 Discussion

The variable assignment weighting scheme presented in Xue, Chan, et al. (2011) was implemented in C language and was compiled by GCC (GNU Compiler Collection). The processes of class association rules learning and weighting assignments were designed as domain-specific and static calls. In comparison, the processes of the presented SPOT algorithm, which aim at robust and cross-domain applications, are designed as open interfaces. For example, the generation of normalized decision attributes on the basis of raw attributes can provide more possible correlations for the models of stochastic machine learning. Another example is the combination of three classification techniques from different branches of supervised learning.

There are several advantages to develop the SPOT algorithm on the HyFlex library. Although the version involved in this thesis is mainly used

for heuristic selection, HyFlex presents a concise and precise infrastructure of interactions between domain-specific LLHs and cross-domain algorithms. Secondly, there are six domains of NP-Complete benchmark problems as well as their LLHs already included in HyFlex. Finally, a public hyper-heuristic competition, CHeSC 2011, already validated the effectiveness and robustness of HyFlex. The competition also attracted tens of hyper-heuristic researchers worldwide and publicized their computational results on HyFlex. In other words, HyFlex provides adequately reliable LLHs and hyper-heuristics for reference.

In this thesis, another hyper-heuristic PHunter is employed as the solver for the subproblem. There were three main reasons. First, as a cross-domain hyper-heuristic, PHunter has fewer structures and codes compared to a number of algorithms in which each one is a specific solver for one domain. Fewer structures and codes also mean fewer possible sources of implementation errors. The second reason is the effectiveness. PHunter was the fourth overall and the first in *hidden* domains[34] out of twenty hyper-heuristics in CHeSC 2011. The last reason is that PHunter and SPOT share the same development library HyFlex. Therefore, it is convenient to incorporate PHunter.

The reason of using a 64-bit Java instead of a 32-bit one lies on the maximum heap size. In a 32-bit Java environment on windows, the maximum heap size is typically 1.8 G bytes (IBM[35] Java) or 4 G bytes (Oracle/Sun Java[36]). However, the class SPOT_ML sometimes needs more memory for storing data and training the classifiers, such as a 3 to 5 G bytes of memory or

[34] See http://www.asap.cs.nott.ac.uk/external/chesc2011/results.html.

[35] See http://publib.boulder.ibm.com/infocenter/javasdk/tools/index.jsp?topic=/com.ibm.java.doc.igaa/_1vg00014884d287-11c3fb28dae-7ff6_1001.html.

[36] According to the "Xmx" parameter.

even more in the large-scale problems in Chapter 5 and Chapter 6. Therefore, a 64-bit Java environment was chosen as the environment.

Chapter 5

Application I: The Traveling Salesman Problem Domain

> Example is the school of mankind, and they will learn at no other.
>
> *Letters on a Regicide Peace*
>
> Edmund Burke

In this Chapter, the presented SPOT hyper-heuristic is applied to a well-known combinatorial optimization domain, the Traveling Salesman Problem (TSP), in order to try to replicate Xue, Chan, et al. (2011)'s modeling approximately. Section 5.1 briefly overviews the background and some well-known algorithms and data structures for TSPs. The implementation of TSP domain in HyFlex is introduced in Section 5.2. The domain-specific design and parameters of the SPOT hyper-heuristic are given in Section 5.3 in detail. Section 5.4 shows the results of experiments and some discussion is given in Section 5.5.

5.1 An Introduction to the TSP Domain

As defined in Section 1.2, a TSP aims at finding the shortest possible tour in which each given city is visited exactly once. Examples of TSPs can be found in Figure 1.6 and Figure A.1. TSP is one of the most comprehensively studied problems in computer science, operational research and engineering practice. A common form of TSPs is *Euclidean* TSP, where the distance between cities is the Euclidean distance. The industrial applications of TSPs include vehicle routing, electric power cable networks, and VLSI (Very-Large-Scale Integration) chip fabrication (Punnen, 2004; Applegate, Bixby, Chvátal, & Cook, 2006).

Both TSP (Garey & D. S. Johnson, 1979) and *Euclidean* TSP (Papadimitriou, 1977) were proven in the NP-Complete class. Papadimitriou and Vempala (2006) showed that TSPs with triangular inequality are NP-hard to approximate, with a ratio less than 220/219. Nevertheless, TSPs, especially *Euclidean* TSPs, can be regarded as well-handled in practice, after decades of research. Algorithms that guarantee finding optimal tours are now capable of solving relatively large-scale instances. For example, the `Concorde` code solved the 85,900 cities in a time of 136 CPU-years for a 2.4 GHz processor (Applegate, Bixby, Chvátal, & Cook, 2006). Besides, efficient heuristics can quickly find tours within a few percent of optimal on real-world instances (D. S. Johnson & McGeoch, 1997). For example, Bentley (1992)'s greedy could find a solution about 15% over the HK lower bound for a TSP with 85,900 cities in three seconds on a 500 MHz EV6 Compaq Alpha processor. More sophisticated heuristics can get within 1% of optima in reasonable amounts

of time (Helsgaun, 2000; D. S. Johnson & McGeoch, 2002).

It is believed that one of the major reasons for the efficiencies of heuristics, especially local search, are the highly effective edge candidate set structures. The main concern of a candidate set is that it is impossible for most of very long edges to appear in optimal tours. So a candidate set usually contains a limited number of *promising* (short or holding particular properties) edges for each city according to the geometry. Popular candidate sets include the *nearest-neighbor-first* (Lin & Kernighan, 1973), the *Delaunay* (Reinelt, 1994), the *k-quadrant* (Miller & Pekny, 1995), and the *α-nearness* (Helsgaun, 2000). There also are other techniques for modifying search procedures according to suboptima, such as the backbone (W. Zhang & Looks, 2005) and the tour-merging (Cook & Seymour, 2003). According to the definitions in this thesis, both the backbone and the tour-merging can be considered as heuristic generation methods in regard to the amount of possible modifications on heuristics, though they did not include complicated and explicit machine learning techniques.

5.2 Implementation of the TSP Domain in HyFlex

The problem domain of TSP was implemented in HyFlex and used as a hidden test domain[37] in CHeSC 2011. The implementation employed ten widely used benchmark instances from the TSPLIB data sets[38], as shown in Table 5.1. All the problems are 2D Euclidean instances. The six drilling instances are from the print circuit board industry, and the two largest in-

[37] Not public before the competition.
[38] Available at: http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/.

Table 5.1: Ten benchmark instances used in the implementation of TSP in HyFlex

No.	Id	Size (n)	Referential optimum	In CHeSC?	Notes
0	pr299	299	48191	✓	Drilling problem
1	pr439	439	107217		Drilling problem
2	rat575	575	6773	✓	Rattled grid
3	u724	724	41910		Drilling problem
4	rat783	783	8806		Rattled grid
5	pcb1173	1173	56892		Drilling problem
6	d1291	1291	50801	✓	Drilling problem
7	u2152	2152	64253	✓	Drilling problem
8	usa13509	13509	19982859	✓	Map of USA
9	d18512	18512	645238		Map of Germany

stances are national maps. The distance implemented in HyFlex was the real-value Euclidean distance. However, the optima in Table 5.1 are based on the rounded integer distance defined in TSPLIB. Because the error caused by the rounding operation is limited, the optima can be used as referential optima to measure the excesses of suboptima approximately. After a random selection, five instances, No. 0, 2, 6, 7 and 8, were selected in the test beds to benchmark the hyper-heuristics in CHeSC 2011.

Twelve LLHs were implemented, as shown in Table 5.2. The LLHs could be categorized into four classes: mutation, ruin-and-recreate, local search and crossover. Algorithms in the crossover class started from two tours and created one new tour for return. Other algorithms started from one tour and return a new (or same) return. All the implemented LLHs were carried out on the nearest-neighbor-first candidate set. Eight nearest neighboring cities were considered as candidates of the preceding or succeeding cities. In other words, two out of eight shortest edges departing from one city were expected to be in the optimum case.

Table 5.2: The LLHs in the implementation of TSP in HyFlex

Type	Id	Parameters used		Notes
		Intensity of mutation	Depth of search	
Mutation	0			Random reinsertion
	1			Swap of two cities
	2			Randomization
	3	✓		Partial randomization
	4	✓		A number of swaps of cities
Ruin-and-recreate	5	✓		Greedy partial reconstruction
Local search	6		✓	First improvement 2-Opt
	7		✓	Steepest descent 2-Opt
	8		✓	First improvement 3-Opt
Crossover	9			Order crossover
	10			Partially mapped crossover
	11			Precedence preservative crossover
	12			One point crossover

5.3 Development of the SPOT for TSPs

This section presents the application of the SPOT heuristic generation for the domain TSP. The development can be divided into the three phases mentioned in Section 3.3:

P1: transformations for standards and sampling,

P2: determination of parameters, and

P3: generation of new LLHs (by modification).

All the domain-specific operations developed in phases P1 and P3 were encapsulated in the interface functions within the class TSP_UEA2, as shown in Figure 4.1. The determined parameters were stored and used by the class SPOT.

5.3.1 P1: Transformations and sampling

The designs of the U/EA and the U/EA^2 standards are described in this section. The first standard is actually the problem model of the SPOT algorithm so that the given combinatorial optimization problems must be transformed to meet the standard. The latter standard regulates the generation of the labeled training data and the unlabeled test data, so that meaningful classifiers can be trained and are compatible with the unlabeled test data.

5.3.1.1 Design and validation of transformations

The hard constraint in a TSP is the requirement for a Hamiltonian cycle. The Hamiltonian cycle is a graph cycle (i.e., closed loop) that visits each node exactly once. In the perspective of a city, the Hamiltonian cycle means there are only *two* edges connected to itself. Alternatively, each city has a degree of two, in a formal description. The only *inter-variable* hard constraint is that all the n edges must form a closed loop.

In the U/EA standard, there is no inter-variable hard constraint. Therefore, the inter-variable hard constraints were directly removed. The inner-variable constraint of "having a degree of two" was kept. Then the transformed problem was actually a relaxed TSP, which could be stated as follows.

> In a transformed TSP, a set $C = \{c_1, c_2, \ldots, c_n\}$ of cities is given, and for each pair (c_i, c_j) of distinct cities there is a distance $d(c_i, c_j)$. The target is to find n pairs so that (i) the summation of the n lengths is *relatively short* and (ii) each city connects two and only

two cities.

It might seem confusing to find "relatively short" targets. In fact, training examples of "relatively short" edges could be obtained from a suboptimum (of a subproblem). The instance-specific information about "what kind of edge seems like 'relatively short' in a similar problem" could be learned, too. The cardinal number of possible assignments for each city was always $n-1$ originally. Henceforth, the new relaxed TSP meets the U/EA standard.

Three instances listed in Table 5.3 were selected to test the *compatibility* of the relaxed TSP in the U/EA standard. The selection was mainly based on the problem size. Small-scale problems, such as those with tens of cities, might bring a biased geographic structure into the learning. Very large-scale problems, such as those with thousands or millions of cities, cost too much time due to the scale. Therefore, three problems with around one thousand cities were selected. In each instance, two arbitrary suboptima were selected randomly to check the resemblance between the two and to estimate the average resemblance of each instance approximately. The approximability, i.e. the overall average resemblance $\bar{r}$, was 70.40%. Both the approximability and generality (degree of approximation among r of instances) were acceptable. Therefore, the U/EA model was said to be compatible to the SPOT hyper-heuristic.

5.3.1.2 Design of decision attributes

The last task in P1 was to design attributes to meet the U/EA^2 standard. The design involved the definitions of the groups of competitors, the raw

Table 5.3: Three benchmark TSP instances used in the development of the SPOT algorithm

No.	Id	Size (n)	% excess the referential optimum		Resemblance r (%)
			Suboptimum 1	Suboptimum 2	
4	rat783	783	0.7609	0.8086	95.27
5	pcb1173	1173	0.5590	0.4394	60.70
6	d1291	1291	2.9279	3.2397	55.23
				Approximability $\bar{r}$ (%)	70.40

attributes and the label(s). The definitions were important. Because both the labeled training data and the unlabeled test data would be generated according to the definitions, and the training data determined the classification and the final LLHs.

The group of competitors for a city c_i was designed as the set of all the $n-1$ edges connecting to the city c_i:

$$\left\{(c_i, c_j) | 1 \leq j \leq n, j \neq i\right\}. \tag{5.1}$$

Each group was marked with a unique identity (integer). In this case, the function `getCompetitiveGroups`() returned an array of $n \times (n-1)$ integers, where each value stood for the identity of an edge group, and there were $p = n - 1$ assignments in each group.

Nevertheless, effective candidate sets were also considered during the design. Given a candidate set, the group for a city c_i became

$$\left\{(c_i, c_j) | c_j \in \text{candidate set of} (c_i, \text{size}_\text{c})\right\}, \tag{5.2}$$

where size_c ($2 \leq \text{size}_\text{c} \leq n-1$) was a constant integer denoting the size of

the edge candidates for each city. The candidate set employed in HyFlex was the nearest-neighbor-first set, and the size_c was 8. In the design of the SPOT hyper-heuristic, the same candidate set was used to generate candidates, and the size_c was set to 20 to include more edges that could possibly be suboptimum assignments. As a result, the implemented function TSP_UEA2.`getCompetitiveGroups`() returned an array of $20 \times n$ integers, where each group consisted of $p = 20$ assignments.

Besides the length of the edges, angular properties were also considered during the design of the attributes, because all the instances were 2D Euclidean TSPs. For each assignment (edge) (c_i, c_j) in each group, there were $ra = 2$ raw attributes designed. The first was the (Euclidean) distance, i.e. $d(c_i, c_j)$. The other raw attribute was the angle of the edge $atan2(y_j - y_i, x_j - x_i)$, where $atan2$ is a variation of the *arctangent* function:

$$atan2(\Delta y, \Delta x) = \begin{cases} \arctan(\Delta y / \Delta x) & \Delta x > 0 \\ \arctan(\Delta y / \Delta x) + \pi & \Delta y \geq 0, \Delta x < 0 \\ \arctan(\Delta y / \Delta x) - \pi & \Delta y < 0, \Delta x < 0 \\ +\pi/2 & \Delta y > 0, \Delta x = 0 \\ -\pi/2 & \Delta y < 0, \Delta x = 0 \\ 0 & \Delta y = 0, \Delta x = 0 \end{cases} \tag{5.3}$$

Therefore, the function TSP_UEA2.`getRawAttributes`() returned a 2D array (in $20 \times n$ rows and 2 columns) of real values. The class SPOT then generated $ra \times (2p + 3) = 86$ columns of normalized attributes and $20 \times n$ rows of data through the function `genNormAtts`(). In fact, the 86 columns of

attributes included 10 of the 14 decision attributes in Xue, Chan, et al. (2011). The tables generated from subproblems were compatible to the unlabeled data in terms of classification. The designed attributes meet the standard of U/EA^2.

The design of the label was, in fact, determined by the design of the group. Because the group was the set of 20 nearest-neighbor-first candidates for each city, the label appeared in the suboptimum for each edge candidate (assignment). The label of each row was set to 1, if the edge appeared in the optimum (or optima); to 0 if not. The labels were obtained through the function TSP_UEA2.`getIntLabels`() which returned an array of $20 \times n$ integer values consisting of $2 \times n$ 1s and $18 \times n$ 0s.

In summary, the transformation and the designs were relatively straightforward and partially used Xue, Chan, et al. (2011)'s model. The objective of the SPOT hyper-heuristic was designed as "generating LLHs to focus on the edge candidates that would likely appear in suboptima". Examples of generated data tables of labeled training data and unlabeled test data can be found in Table B.1 and Table B.4 in Appendix B.

5.3.2 P2: Parameter determination

There were two important parameters to be determined, one by one, in this phase, as described in Section 3.3.1. The first parameter was the size of the subproblems n', which was numeric. The other was the method of sampling for generating subproblems, and it was non-numeric.

5.3.2.1 The size n' of subproblems

The tests of n' were set up as follows. The function SPOT.`run`() was executed without the final step of generating the LLHs. It was because the indicators were measurable before the generation of LLHs. The sampling method was set to the *random selection*. The attribute selection was enabled by calling the function `trainModelAndTestAfterSel`() for classification. The usual number of selected attributes was around ten, see Table B.1 and Table B.3 in Appendix B for a comparison. The computation time of the embedded subproblem solver PHunter was set to six seconds. After training the classifiers and labeling the test data, the 20-nearest-neighbor-first candidate sets were sorted by the *weighted distance* suggested by Xue, Chan, et al. (2011):

$$\left(1 - \frac{\gamma_{idx}}{3}\right) \times d(c_i, c_j), \tag{5.4}$$

where c_j was the idx-th edge in the candidate set of c_i and γ_{idx} was the summation of predicted labels for the assignment (c_i, c_j). Then size_c was set to 8 for generating a comparable data structure. In fact, weighting the edge candidates is not a new idea, and relevant examples could be can in different heuristic and exact algorithms, such as Nemhauser and Wolsey (1988, pp. 494–495)'s branch-and-bound tree.

The indicator r was decomposed into two indicators to measure the suboptimality of the generated candidate set. The first indicator cov was the average *coverage of suboptimum assignments*, which was the percentage of the suboptimum assignments covered by the candidate set. It is clear that

$0 \leq cov \leq 1$ and a greater cov usually stood for a more expected[39] result for learning and a better suboptimality. The other indicator $\bar{d}$ was the *average depth*, which measured the average index of the optimum assignments in the candidate set. In other words, $\bar{d}$ showed how many non-suboptimum assignments were "wrongly" placed before suboptimum ones, on average. A lower level of $\bar{d}$ denoted a better accessibility of the promising assignments when the cov was not changed.

Six values of n' were tested, as shown in Figure 5.1. One hundred independent tests were carried out for each value and each instance. Figure 5.1 shows the trends of the average coverage cov, the average depth $\bar{d}$ and the time cost against the value of n'. The "8-nearest-neighbor-first" was the data set implemented in HyFlex.

It can be seen from Figure 5.1 that the new candidate set included more suboptimum assignments when $n' \geq 50$. A decreasing trend of average depth $\bar{d}$ meant that the suboptimum assignments were gradually moving towards the heads of the ordered groups when n' increased. Comparing to the candidate set implemented in HyFlex, the new candidate set generally covered more suboptimum assignments but placed them slightly deeper. For instance, when $n' = 800$ the new candidate set covered $+0.142\%$[40] more on the fraction but placed the suboptimum assignments $+0.017$ deeper when $n' = 800$. It was acceptable, because cov, which related to the scope of search of LLHs, was slightly more important than $\bar{d}$ which related to the depth of search.

[39] A $cov \approx 1$ could also possibly mean the learning was overfitting in some cases.
[40] See Table C.2 for data.

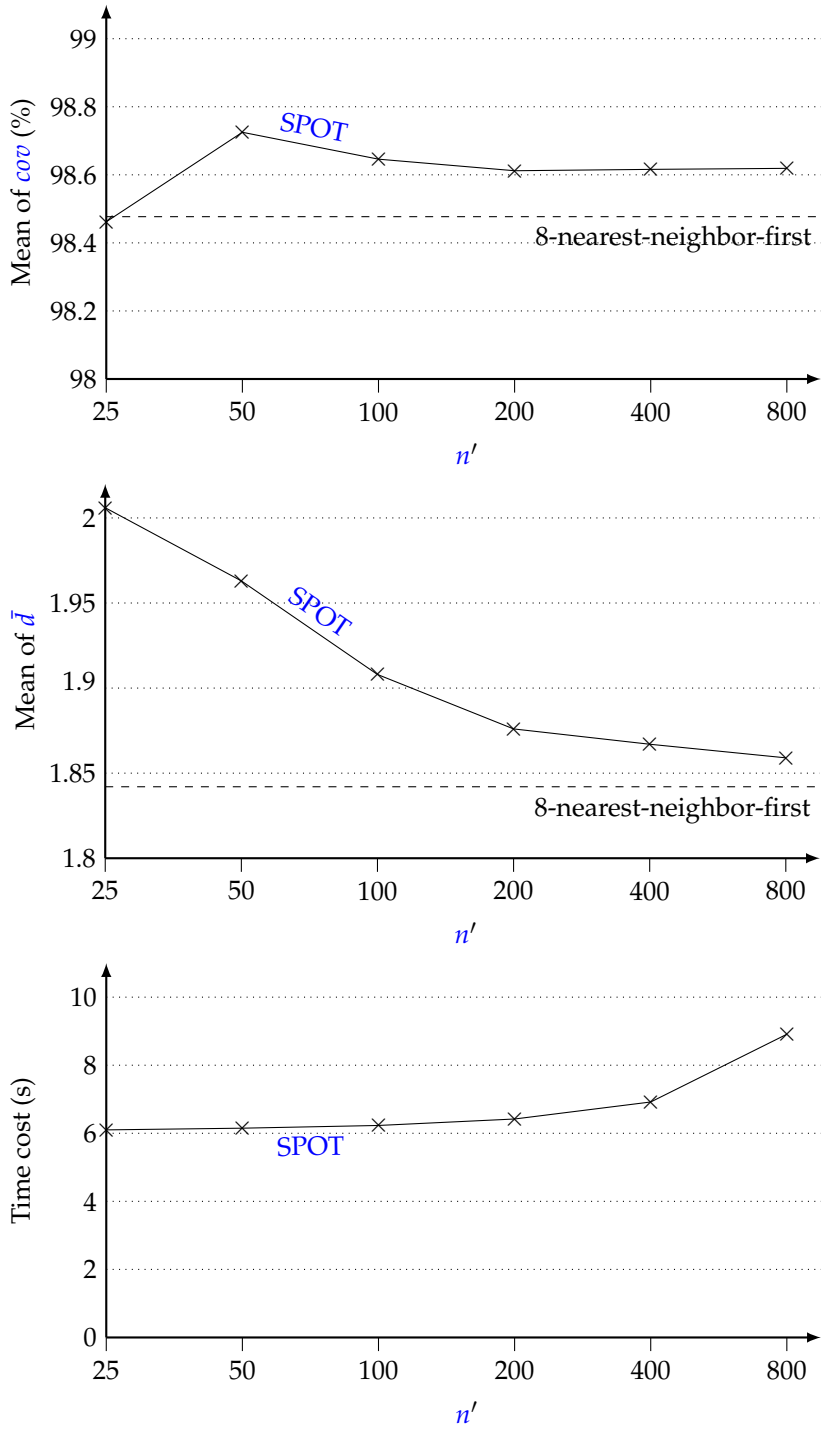


Figure 5.1: Average test results for determining n' in the development of the SPOT algorithm, on the three TSP test instances (100 runs)

The size n' was chosen as 400 after the tests. The main reason was to balance performance and time cost. The time spent on learning seemed to increase considerably when $n' > 400$. If too much time was spent on learning, it would be difficult to call the application an "on-the-fly" hyper-heuristic.

5.3.2.2 The methods of sampling for subproblems

Besides the general random selection, another method, *rectangular selection*, was also tested to try to fine-tune the SPOT algorithm. For the 2D Euclidean instances in HyFlex, the geography of the cities can be regarded as being in a minimum boundary rectangle. A smaller rectangle can be *selected* within the boundary rectangle to choose a block (subset) of cities. Given the size n of the input problem and the size n' of the subproblems, the area of the small rectangle is n'/n of that of the minimum boundary rectangle, if the cities are assumed to be distributed uniformly in space. In practice, a few small rectangles can be randomly placed and the one with the closest density of cities can be selected, in order to reduce errors from possible irregular distribution of cities.

The *aspect ratio* of the small rectangle was considered as a parameter. Letting the aspect ratio of the minimum boundary rectangle be a constant 1:1, five ratios were tested, as shown in the first column in Table 5.4. It can be seen that both the best (greatest) cov and the best (least) $\bar{d}$ were generated in the ratio 1:1, which meant keeping the same aspect ratio as the minimum boundary rectangle. Therefore, the aspect ratio in the rectangular selection was set to be the same as the boundary box in the following.

Tests were carried out to examine what sampling method was able to

Table 5.4: Average test results on different aspect ratios in the rectangular selection sampling method, where $n' = 400$ and the aspect ratio of the minimum boundary rectangle is assumed as 1:1 (100 runs, best performance in bold)

Aspect ratio	cov (%)	$\bar{d}$	$\text{Time}_{\text{overall}}$ (s)	Time_{L} (s)
1:16	98.570	1.936	6.30	0.28
1:4	98.613	1.874	6.65	0.61
1:1	**98.615**	**1.869**	6.94	0.89
4:1	98.612	1.879	6.66	0.62
16:1	98.563	1.876	6.31	0.29

generate a sample that led to the best candidate set in terms of the two indicators. Table 5.5 shows the average results and significance by one-way ANOVA tests[41]. The first column of Table 5.5 is the indicator, and the second column denotes the test instance. Results of the rectangular selection, the random selection and two parallel subproblems where each method sampled one of them are, listed in the third column, the fourth column and the fifth columns, respectively. Given the method of sampling as categories, the F values and the significance p show the variances of mean values and the statistical significance of null hypothesis, as listed in the last two columns, respectively.

It can be observed from Table 5.5 that the average values on cov were not always significant, though the cov resulted by the "Parallel" column seems slightly higher. The average values on $\bar{d}$ were clearly significant, as written in bold. Therefore, it was statistically significant that the $\bar{d}$ relating to the parallel runs of both selections was the best (least). The time spent on the parallel mode was approximately twice much as that spent on one

[41]The one-way ANOVA (analysis of variance) is employed to compare means of two or three samples in this thesis. The null hypothesis is that two or more groups are drawn from the same population. When there are only two means to compare, the relation between ANOVA and the t-test is given by $F = t^2$.

Table 5.5: Average results and significants of tests on different subproblems sampled, where $n' = 400$ (100 runs, significant p in bold)

Indicator	Instance	Rectangular selection	Random selection	Parallel	ANOVA	
					F	p
cov	4	99.617	99.617	99.617	0.51	0.60
	5	98.892	98.892	98.892	—†	—
	6	97.332	97.343	97.345	3.12	**0.05**
	Average	98.613	98.617	98.618		—
$\bar{d}$	4	2.103	2.107	2.103	6.65	**0.00**
	5	1.893	1.906	1.895	92.79	**0.00**
	6	1.652	1.618	1.599	19.80	**0.00**
	Average	1.883	1.877	1.866		**0.00**

†: No variance.

subproblem. In the following, two parallel subproblems were sampled by random selection and rectangular selection, respectively. Each subproblem and its suboptimum contributed half of the training data for supervised learning.

With the determined transformations, design of attributes, sampling and other secondary algorithms and parameters, a predicted label could be generated, and a weighted distance could be calculated for each assignment (in the candidate set) on-the-fly. However, it should be noted that the 8-nearest-neighbor-first candidate set already achieved an effective pair of indicators, $cov = 98.477\%$ and $\bar{d} = 1.842$, as shown in Figure 5.1. The average results in the "Parallel" column in Table 5.5 only shows a slightly higher coverage ($\Delta cov = +0.141\%$) but a slightly weakened depth ($\Delta\bar{d} = +0.024$). Therefore, the new candidate set, as well as newly generated heuristics, should be capable of providing slightly higher qualified solutions, while it might need in-depth search capability to find them.

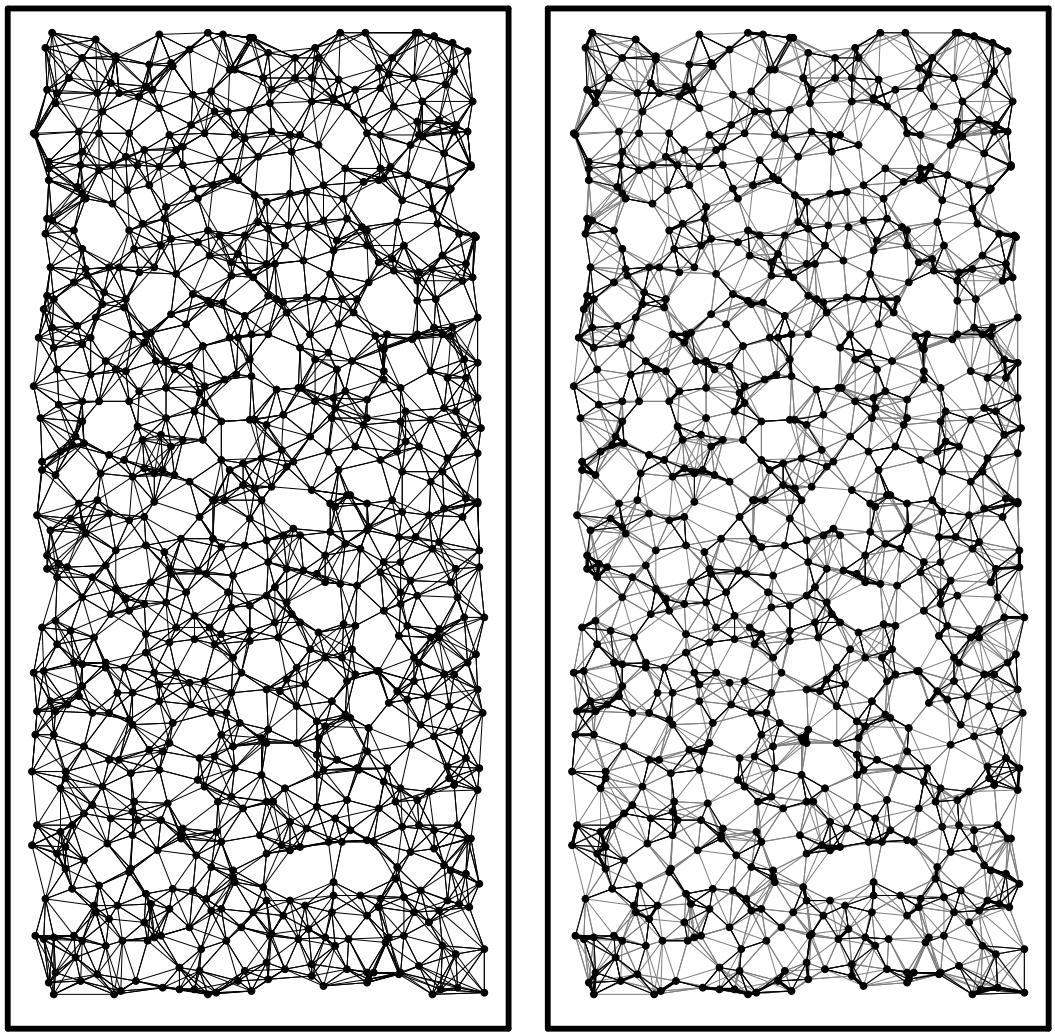
5.3.3 P3: The generation of new LLHs

This part was the last mile of the development and of the function TSP_UEA2.`setIntLabels`(). The new candidate set was obtained after P1 and P2. Figure 5.2 shows an example of the new candidate set. In the second figure in Figure 5.2, thick lines are the assignments with Σ labels ≥ 2, the thin lines are those with Σ labels $= 1$ and the gray lines are those with Σ labels $= 0$. Although there were priorities given in the new candidate set, the difference between the two figures in Figure 5.2 is actually not significant, as indicated by *cov* and $\bar{d}$. More details of generating the example can be found in Appendix B.

The new candidate set generated by the SPOT algorithm was *only* applied to the three local search LLHs in Table 5.2. The reason was the local search LLHs had stronger in-depth search capabilities than those in other categories. The algorithms number 6, 7 and 8 listed in Table 5.2 were called number 6', 7' and 8' after change of the candidate set. Although the codes of all local search LLHs remained the same, the scope and traces of their exploration of the search space changed due to the change of the candidate set. The original algorithm numbers 6, 7 and 8 were called "base algorithms" of the new ones. Generally, the three new LLHs can be regarded as *generated* by the SPOT hyper-heuristic.

5.4 Experiments and Observations

This section presents two groups of tests. In the first group of experiments, the individual performance of each new LLH was examined, with



(i) The 8-nearest-neighbor-first (ii) A new candidate set by the SPOT

Figure 5.2: A comparison of two candidate sets of the TSP instance `rat783`

comparisons to its base algorithm. In the other group of tests, the three new LLHs replaced their base algorithms. The heuristic selection method PHunter solved the updated domain consisting of new LLHs. The results were compared to the solving of PHunter with the original LLHs and the winners in the TSP domain in CHeSC 2011.

All the tests were carried out with the same instances which had been selected for the competition in CHeSC 2011, i.e. the numbers 0, 2, 6, 7 and

8 shown in Table 5.1. The size n' of the subproblems was set to 400 and two subproblems were generated in parallel to provide training data, as described in Section 5.3.2. The overall computation time was set to be the same as the normalized time in CHeSC 2011, which was equivalent to 10 CPU minutes for a Pentium 4 3.0 GHz computer. All the time costs of the processes in the SPOT hyper-heuristic, such as sampling subproblems, solving the subproblems, learning instance-specific knowledge and modifying LLHs, were fully included in the overall time in order to make the results comparable. For the embedded subproblem solver PHunter, the time limit for solving each subproblem was 1% of the overall time, i.e. equivalent to 6 CPU seconds on a Pentium 4 3.0 GHz machine. The time of the rest processes in the SPOT was not directly restricted.

5.4.1 On the individual LLHs

Table 5.6 shows the results on *tour quality* of the newly generated LLHs in terms of percentage excess over the referential optima[42]. There were 100 runs for each configuration and each problem instance. In each run, a random initial tour was generated for all local search LLHs and all LLHs started from the same initial tour independently. One-way ANOVA tests were also included to check the significance of the changes. The second column is the test instance. The third and the fourth columns show the excesses of the shortest tour length and the average length found by the base algorithms in HyFlex, respectively. The fifth and the sixth columns stand for the shortest tour length and the average length found by the new LLHs,

[42]It should be noted that the referential optima do not necessarily present the true optima.

Table 5.6: Test results on the tour quality of the generated LLHs and their base algorithms (100 runs, $n = 400$, Depth of search $= 1$, improvements on average performance in bold)

Group	Inst	% excess by the base LLH		% excess by the new LLH		Δ avg	ANOVA	
		Min	Avg	Min	Avg		F	p
No. 6	0	6.944	12.941	6.944	12.993	0.052	0.01	0.93
versus	2	6.329	8.151	6.329	8.156	0.005	0.00	0.97
No. 6′	6	9.690	17.066	9.900	17.172	0.106	0.05	0.83
	7	10.905	14.679	10.729	14.660	**-0.019**	0.01	0.93
	8	10.027	11.840	10.027	11.840	0.000	0.00	—
			Avg improvement (%)			-0.190	Avg	0.92
No. 7	0	6.273	12.271	5.909	12.107	**-0.164**	0.07	0.79
versus	2	5.915	8.560	5.915	8.559	**-0.001**	0.00	0.99
No. 7′	6	9.683	16.750	10.038	16.900	0.150	0.05	0.76
	7	10.957	14.407	10.707	14.240	**-0.166**	0.05	0.44
	8	9.515	11.589	9.515	11.589	0.000	0.00	—
			Avg improvement (%)			**0.322**	Avg	0.75
No. 8	0	3.548	7.164	2.535	7.045	**-0.119**	0.11	0.74
versus	2	3.113	4.301	3.113	4.292	**-0.009**	0.01	0.91
No. 8′	6	6.657	14.083	6.683	13.984	**-0.099**	0.04	0.84
	7	7.346	10.507	7.108	10.387	**-0.120**	0.31	0.58
	8	5.844	6.985	5.844	6.985	0.000	0.00	—
			Avg improvement (%)			**0.743**	Avg	0.77

respectively. The seventh column denotes the change on the average excess, where improvements (negative numbers) are written in bold. The last two columns present the F values and the confidence p examined by the one-way ANOVA tests. The last row in each group shows the average improvement in percentage and the mean of p in each group. The average improvement of each new LLH is defined as

$$\text{Avg improvement} = -\frac{\text{new \% excess} - \text{base \% excess}}{\text{base \% excess}} \times 100\%. \quad (5.5)$$

It can be observed from the seventh column that the generated No. 7′ and

No. 8′ LLHs found slightly better ($0 <$ average improvement $< 1\%$) tours on average than their base algorithms. However, the generated No. 6′ local search returned many inferior (average improvement < 0) results on average, compared to its base algorithm. The improvements by No. 8′ seems more steady (lower variance in the seventh column) and slightly more in amount (+0.743% against +0.322%) compared to those of No. 7″s. However, all the changes, no matter whether improved or weakened, were not statistically significant[43] ($p \gg 0.05$).

Nevertheless, the observations are consistent with the interpretation and prediction according to the indicators *cov* and $\bar{d}$ in Section 5.3.2. In fact, No. 6 LLH was the first improvement 2-Opt and No. 7 LLH was the steepest descent 2-Opt, as shown in Table 5.2. The only difference between them was that the first one accepts any swap of two cities if it can improve the tour and the second one only accepts the swap with best improvement. Hence, it can be observed that the differences between the first and the second groups in the seventh column in Table 5.6 were considerable. No. 8 LLH was the first improvement 3-Opt, which explored a higher level of swaps than No. 6. The observation of the differences between the corresponding groups in Table 5.6 verified that in-depth search algorithms were slightly better in this particular application of the SPOT heuristic generation. In fact, Xue, Chan, et al. (2011) also observed a similar phenomenon that an α-nearness-like candidate set determined by machine learning made obvious improvements on a 5-Opt local search but encountered significant failure on

[43] It should be noted that the insignificant and slight improvements did not necessarily lead to the conclusion that the SPOT algorithm was useless in the TSP domain. In fact, the same implementation could generate much more powerful LLHs by changing the rule of sorting edge candidates. See Section 5.5 for more details.

Table 5.7: Average time of generating new LLHs (100 runs, $n = 400$, Depth of search = 1)

Instance	Size n	Time (s)			
		Min	Avg	Max	Std dev
0	299	12.70	14.10	16.75	0.66
2	575	13.95	14.92	17.41	0.59
6	1291	13.10	14.35	16.43	0.48
7	2152	12.95	13.49	14.66	0.32
8	13509	19.47	23.03	28.69	2.25

a 2-Opt heuristic for Euclidean TSPs.

The *time cost* of *generating* the LLHs is shown in Table 5.7. The third to the sixth column show the minimum, the average, the maximum values and the standard deviation of the time cost of the whole SPOT hyper-heuristic for each instance. It should be noted that a 12 equivalent CPU seconds for solving two subproblems was included in the data. It could be observed that the maximum time cost in the instance with 13509 cities was no more than about 30 seconds. Therefore, the SPOT could facilitate an instance-specific heuristic generation in an "on-the-fly" fashion.

Table 5.8 shows the run time of the LLHs in solving problems in milliseconds. The computation time of No. 6, No. 7 and No. 8 LLHs which was provided by HyFlex is shown in the second, the fifth and the eighth columns, respectively. The time of No. 6', No. 7' and No. 8' LLHs generated is shown in the third, the sixth and the ninth columns, respectively. The fourth, the seventh and the last columns show the changes of average time in the three groups. It can be observed that the changes of time in the first four instances were almost unnoticeable, especially when referring to the level of the instrument error. In the last row, No. 7' and No. 8' LLHs took

Table 5.8: Average run time of each LLHs (100 runs, $n = 400$, Depth of search = 1, instrument error < 0.16 ms)

Instance	Average run time of LLHs in the three group (ms)								
	No. 6	No. 6′	Δ	No. 7	No. 7′	Δ	No. 8	No. 8′	Δ
0	0.6	0.6	0.0	0.5	0.5	0.0	8.7	8.4	-0.3
2	1.3	1.3	0.0	1.4	1.3	-0.2	15.9	16.2	0.3
6	3.0	2.8	-0.2	2.8	2.8	0.0	52.7	53.5	0.8
7	6.2	6.2	0.0	6.2	6.6	0.3	111.1	111.1	0.0
8	103.6	97.2	-6.4	100.8	104.4	3.6	4162.9	4191.6	28.7

about 4% and 1% more time than their base algorithms, respectively, and No. 6′ one saved about 6% of time. In general, the computation time of the new LLHs was not considerably different from those of their base algorithms. With consideration of the tour qualities of the new LLHs, No. 7′ and No. 8′ could possibly replace their base algorithms in the HyFlex framework.

5.4.2 Comparisons with other hyper-heuristics

Many existing hyper-heuristics, especially heuristic selection methods, can benefit from the newly generated LLHs by the SPOT hyper-heuristic by employing the new LLHs into their LLH sets. For example, the generated instance-specific LLHs, especially No. 7′ and No. 8′, could replace their base algorithms in HyFlex. The heuristic selection methods, such as the PHunter, can run on the basis of an updated LLH set. Such a combination of SPOT preprocessor and the body of the PHunter is noted as the SPOT-PHunter in the following.

The HyFlex framework and the CHeSC 2011 competition provided public and reliable results[44] for twenty hyper-heuristics. In the TSP domain in

[44]See http://www.asap.cs.nott.ac.uk/external/chesc2011/raw-results.html.

Table 5.9: The top three hyper-heuristics in the TSP domain in CHeSC 2011

No	Algorithm	Score	Author
1	AdaptHH	40.25	Mısır et al. (2012)
2	EPH	36.25	David Meignan
3	PHunter	26.25	Chan et al. (2012)

CHeSC 2011, the top three hyper-heuristics are listed in Table 5.9. The scoring system[45] was the Formula one points system, in which the winner got 10 points, the runner-up got 8 points, and the third got 6 in each instance. There were five instances selected. Therefore, the maximum of possible score was 50.

The SPOT-PHunter was tested as a normal algorithm player in CHeSC 2011. The parameters of the PHunter in the body were the same as determined in Chan et al. (2012). All the three new local search LLHs were used to replace their base algorithms, even though No. 6′ might be less competitive than its base algorithm. The tests were conducted for the five instances, and each was tested 31 times. The time cost in each test was equivalent to 10 minutes on a Pentium 4 3.0 GHz CPU. The median values[46] were used for scoring, as shown in Table 5.10. The second to the fourth columns show the results of the top three hyper-heuristic approaches in CHeSC 2011. The fifth and the sixth columns present the median solutions found by PHunter and the SPOT-PHunter. Because the tests in CHeSC 2011 were carried out on a 32-bit[47] Java environment, the third and the fifth columns of PHunter data in the table were labeled as $\text{PHunter}_{32\text{bit}}$ and $\text{PHunter}_{64\text{bit}}$, respectively. The last column shows the confidence of one-way ANOVA test on comparing the

[45] See http://www.asap.cs.nott.ac.uk/external/chesc2011/scoring.html.
[46] The median value of a set $\{1, 1, 3\}$ is 1, while the average value is 5/3.
[47] Confirmed by private communication in 2011.

Table 5.10: Median tour length of the PHunter, the SPOT-PHunter and three other hyper-heuristics in the TSP domain in HyFlex (31 runs, best results in bold for each instance)

Inst	Best results† in CHeSC 2011			Test results in thesis		
	AdaptHH	EPH	PHunter_{32bit}	PHunter_{64bit}	SPOT-PHunter	ANOVA p
0	**48194.9**	**48194.9**	**48194.9**	**48194.9**	**48194.9**	—‡
2	6810.5	6811.9	6813.6	6812.6	**6802.4**	0.00
6	53099.8	52925.3	52934.4	**52835.2**	52872.7	0.28
7	66879.8	66756.2	67136.8	66826.0	**66606.4**	0.00
8	**20822145.7**	21064606.3	21246427.7	21221065.6	21419868.0	0.00

†: Results on a 32-bit Java.
‡: Exactly the same.

mean tour qualities between PHunter_{64bit} and SPOT-PHunter. The best results for each instance are written in bold. Two entries, the PHunter_{64bit} and the SPOT-PHunter, were added to the set of twenty hyper-heuristics. The updated scores, according to the scoring system, of the top five hyper-heuristics are shown in Figure 5.3.

First, it should be noted that the median tour length returned by the PHunter on a 64bit Java environment was equal to or slightly better (less) than that on a 32bit Java in *every* test instance, as shown in Table 5.10, though the codes (and complied binaries) of the PHunter algorithm remained exactly the same. Therefore, the results of the PHunter_{64bit} and the SPOT-PHunter are more comparable.

The involving of new LLHs generated by the SPOT increased the score of the PHunter from 27 to 34, as shown in Figure 5.3. Besides No. 0 instance in which every algorithm returned identical results, the SPOT-PHunter beat the PHunter_{64bit} in two instances (No. 2 and No. 7), and both improvements are significant. The PHunter_{64bit} obtained better results in two other instances, and one improvement is significant and another is insignificant

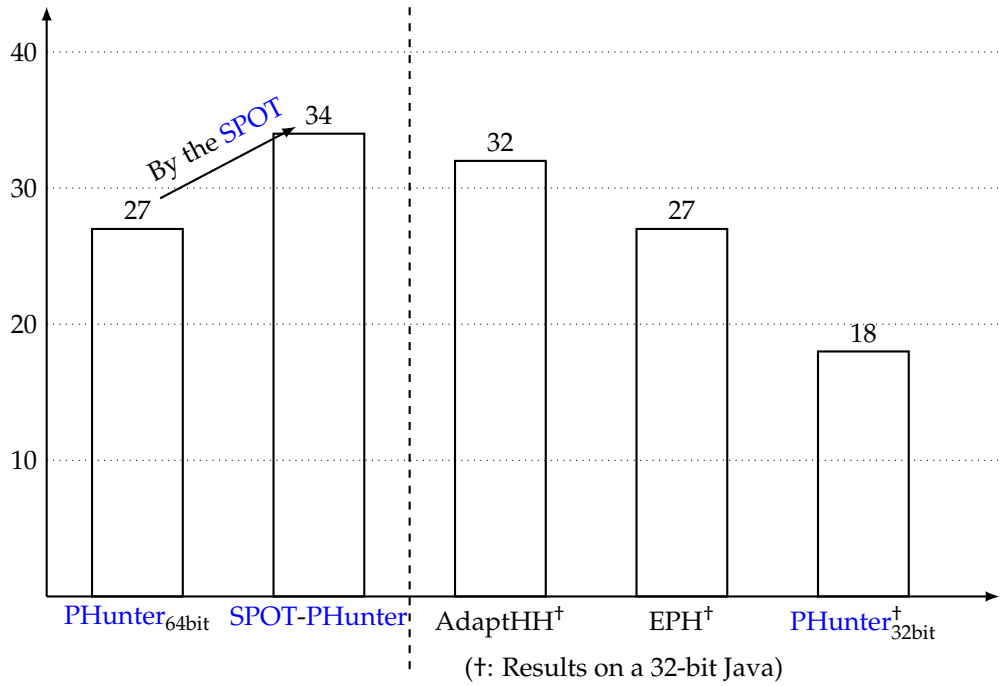


Figure 5.3: Scores of the PHunter, the SPOT-PHunter and three other hyper-heuristics in the TSP domain in HyFlex

($p = 0.28 > 0.05$). In summary, it can be observed that the SPOT hyper-heuristic insignificantly and slightly improved the PHunter algorithm. The result is consistent with the observation and interpretation on the insignificant and slight improvements in Section 5.4.1. What made the SPOT-PHunter have better scores, as shown in Figure 5.3, was the scoring system. For example, in the instance No. 8 in Table 5.10, the $\text{PHunter}_{\text{64bit}}$ did not gain much advantage in scoring after beating the SPOT-PHunter. Its score in this instance was low, because it was still dominated by the AdaptHH and the EPH.

5.5 Discussion

Compared to the algorithm in Xue, Chan, et al. (2011), the SPOT heuristic generation implemented in this study was only able to make insignificant and slight improvements in both theory and practice. Although the observations *partially verified* the theoretical prediction by the indicators, there were certain reasons that led to the insignificant and slight improvements. One main reason is related to the capability of in-depth search of the local search LLHs for modifying. For example, one of the major LLHs for modifying was an efficient 5-Opt in Xue, Chan, et al. (2011), and the base LLHs in this study were 3-Opt and 2-Opt local search. Another possible reason was the size of the candidate set. When the size was 8 in HyFlex, the indicator *cov* showed that a 8-nearest-neighbor-first candidate set had an almost 99% coverage of suboptimum assignments. That meant there was very limited room for the SPOT hyper-heuristic to improve. In comparison, the coverage of the 5-nearest-neighbor-first candidate set was about 97% and the maximum room to improve was then about 3%.

The HyFlex is a great platform to test new hyper-heuristics. It is open and flexible, and encapsulates a lot of algorithms and data structures in its implementation. Nevertheless, there is still some room to extend and/or to improve its LLHs and data structures further. For example, the 2-Opt in HyFlex returned tours with more than a 13% excess over the referential optima on average, as shown in Table 5.6, when the 2-Opt local search in Helsgaun (2000) could find tours within a 4% excess, in general, in the same data set of benchmark instances. Some other candidate sets could also be

considered. For example, according to experiments, a k-quadrant candidate set noticeably reduced the average excess of tours by a considerable 35%. The same implementation of the SPOT hyper-heuristic generated new LLHs, that further reduced the average excess of the k-quadrant candidate set by around 1%, by sorting the edges in the k-quadrant order. The SPOT heuristic generation in this study can probably generate stronger LLHs in an updated implementation of the TSP domain.

Chapter 6

Application II: The Permutation Flow-Shop Scheduling Problem Domain

μεταβολὴ δὲ πάντων γλυκύ.

Change in all things is sweet.

The Nicomachean Ethics

Aristotle

This chapter presents an application of the SPOT heuristic generation for another well-known combinatorial optimization domain, the permutation Flow-Shop scheduling Problem (FSP). The structure of this chapter is similar to that of Chapter 5. A brief introduction to FSPs is given in Section 6.1, and Section 6.2 describes the implementation of the FSP domain in HyFlex. The adaptation and design of the SPOT heuristic generation are presented in Section 6.3. Experiments and observations are shown in Section 6.4, concluding with general discussion in Section 6.5.

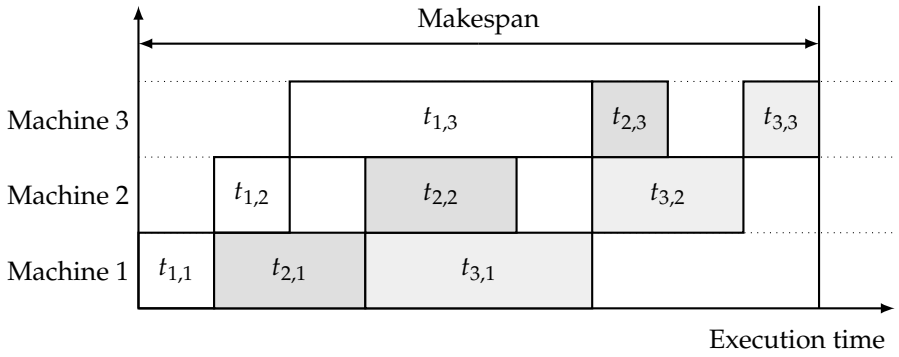


Figure 6.1: An example of an FSP schedule with a permutation (J_1, J_2, J_3)

6.1 An Introduction to the FSP Domain

FSP, sometimes referenced as $F/permu/C_{\max}$ or $n/m/p/C_{\max}$ notationally, aims at finding a permutation of n jobs that minimizes the makespan. Figure 6.1 shows an example of an FSP schedule. FSP has been comprehensively studied since the 1950s and still attracts many researchers even now, see S. M. Johnson (1954), Heller (1960), Ignall and Schrage (1965), Taillard (1993), Reza Hejazi and Saghafian (2005). FSP and its variants can be applied to the traditional domains such as the surface mount technology and printed circuit manufacturing as well as some innovative domains such as multimedia object scheduling in the World Wide Web (Allahverdi & Al-Anzi, 2002).

As stated in Chapter 1, FSP was proven to be NP-Complete (Garey, D. S. Johnson, & Sethi, 1976) and NP-hard to approximate with a factor less than 5/4 (Williamson et al., 1997). However, S. M. Johnson (1954)'s polynomial-time two-machine scheduling had a strong influence in the later algorithms, such as Ignall and Schrage (1965)'s Branch-and-Bound, Campbell, Dudek, and Smith (1970)'s CDS and Gupta (1975)'s approach. Similar to the TSP

domain, there are effective heuristics for FSPs, too. A well-known heuristic is Nawaz et al. (1983)'s NEH heuristic which employs a "curtailed-enumeration" and can obtain very satisfactory solutions — usually within a 5% excess over the best-known solutions in large-scale FSPs. Other exact and heuristic algorithms, such as Reeves (1995)'s genetic algorithm and Taillard (1990)'s tabu search, are also popular in solving FSPs. Some hyper-heuristic research has also been carried out. For example, Vázquez-Rodríguez and Ochoa (2011) investigated evolving ranking functions for NEH to prioritize operations.

6.2 Implementation of the FSP Domain in HyFlex

HyFlex implemented the FSP domain and publicized a part of the FSP instances for open tests in CHeSC 2011. There were twelve FSP instances, as shown in Table 6.1. The instances were selected from a well-studied benchmark data set[48] by Taillard (1993). All the problems were randomly generated instances. The best-known solutions in the fourth column in Table 6.1 were collected from Zobolas, Tarantilis, and Ioannou (2009). Three instances, Nos. 1, 3 and 8, were randomly selected into the test beds. Two more instances, Nos. 10 and 11, which were not publicized before the competition, were also included for evaluating the hyper-heuristics in CHeSC 2011.

Table 6.2 shows fifteen LLHs implemented in FSP domain. The LLHs could also be categorized into the four classes: mutation, ruin-and-recreate, local search and crossover. Similar to the crossover methods for TSPs, crossover algorithms for FSPs started from two tours and created one new

[48] Available at: http://mistic.heig-vd.ch/taillard/problemes.dir/ordonnancement.dir/ordonnancement.html.

Table 6.1: Twelve benchmark instances used in the implementation of FSP in HyFlex

No.	Id	Size ($n \times m$)	Best known	In CHeSC?	Notes
0	TA081	100×20	6202		Randomly generated
1	TA082	100×20	6183	✓	Randomly generated
2	TA083	100×20	6271		Randomly generated
3	TA084	100×20	6269	✓	Randomly generated
4	TA085	100×20	6314		Randomly generated
5	TA092	200×10	10480		Randomly generated
6	TA093	200×10	10922		Randomly generated
7	TA111	500×20	26059		Randomly generated
8	TA112	500×20	26520	✓	Randomly generated
9	TA114	500×20	26456		Randomly generated
10	TA101	200×20	11195	✓	Randomly generated
11	TA113	500×20	26371	✓	Randomly generated

Table 6.2: The LLHs in the implementation of FSP in HyFlex

Type	Id	Parameters used		Notes
		Intensity of mutation	Depth of search	
Mutation	0			Random reinsertion
	1			Swap of two jobs
	2			Randomization
	3	✓		Partial randomization
	4			The NEH heuristic (Nawaz et al., 1983)
Ruin-and-recreate	5	✓		Partial reconstruction by the NEH
	6	✓	✓	Partial reconstruction by a greedy stepper NEH
Local search	7			Iterated steepest descent 1-job interchange
	8			Iterated first improvement 1-job interchange
	9		✓	Random steepest descent 1-job interchange
	10		✓	Random first improvement 1-job interchange
Crossover	11			Order crossover
	12			Precedence preservative crossover
	13			Partially mapped crossover
	14			One point crossover

solution. Other algorithms started from one tour and returned a new (or same) one. Many LLHs were based on the effective NEH heuristic which involved iterations of assignment trials (limited backtracking).

6.3 The Development of a SPOT for the FSP

This section presents an application of the SPOT hyper-heuristic for the domain FSP, in the form of the three development phases mentioned in Section 3.3:

P1: transformations for standards and sampling,

P2: determination of parameters, and

P3: generation of new LLHs (by modification).

All the domain-specific operations developed in phases P1 and P3 were encapsulated in the interface functions within the class FSP_UEA2, as shown in Figure 4.1. The determined parameters were stored and used by the class SPOT.

6.3.1 P1: Transformations and sampling

The designs of the U/EA and the U/EA^2 standards are described in this section. The first standard focuses on the input problem. Transformations between permutations of FSPs and 0-1 matrices are introduced to meet the U/EA standard. The latter standard aims at delivering meaningful and sufficient information in terms of attributes, which can be partially borrowed from existing algorithms.

6.3.1.1 Design and validation of transformations

Every solution of an FSP is a unique permutation (or sequence) of n jobs. The size of the search space is $n!$. When n is a large number, the size can be approximated by Stirling's formula[49]:

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n e^{\lambda_n}, \tag{6.1}$$

where $\frac{1}{12n+1} < \lambda_n < \frac{1}{12n}$. Given an FSP, a Boolean function of the relationship "following" between two jobs J_i and J_j can be defined as follows:

$$fl(i,j) = \begin{cases} 0 & \text{iff. index of } J_i \leq \text{ index of } J_j, \\ 1 & \text{otherwise.} \end{cases} \tag{6.2}$$

Any permutation of n jobs can be transformed to an $n \times n$ 0-1 (or Boolean) matrix, where each element is the value of function fl on its row index and column index. For example, a permutation (J_3, J_2, J_1) can be written as the matrix

$$\begin{bmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \tag{6.3}$$

The conversion from a matrix to a permutation can be stated as follows. Let the summation $\sum_{1 \leq j \leq n} fl(i,j)$ of the i-th row be the weight of the i-th job J_i. The permutation refers to the ordered jobs which are sorted by the weights in the ascending order.

[49] See http://en.wikipedia.org/wiki/Stirling's_approximation.

It is clear that $fl(i,i) = 0$ must be a constant for any i. In a designed transformed FSP, each $fl(i,j)$ is considered as a variable, where $i \neq j$. Therefore, there are $n^2 - n$ Boolean variables. The new transformed problem can be described as follows.

> In a transformed FSP, n jobs are given to be processed on m machines in the same order, the process time of job i on machine j being $t_{i,j}$ ($i = 1,2,\ldots,n$; $j = 1,2,\ldots,m$). A 0 or 1 value of "following" needs to be determined for each pair of jobs such that the total elapsed time (makespan) is minimized.

The variables are unconstrained, and each variable is Boolean. Henceforth, the transformed FSP meet the U/EA standard. The cardinal number of the new search space is 2^{n^2-n}. The new search space is, actually, much larger than the original search space when n is a large number, for example,

$$\begin{aligned} \lim_{x\to+\infty} \frac{2^{n^2-n}}{n!} &= \lim_{x\to+\infty} \frac{2^{n^2-n}}{\sqrt{2\pi n}\left(\frac{n}{e}\right)^n} \\ &= \lim_{x\to+\infty} e^{\ln 2\left(n^2-n\right) - \frac{\ln 2+\ln \pi+\ln n}{2} - n(\ln n - 1)} \\ &= \lim_{x\to+\infty} e^{\ln 2 n^2} \\ &= +\infty. \end{aligned} \tag{6.4}$$

Therefore, the transformation from permutations to matrices is a one-to-one mapping and the transformation from matrices to permutations is a many-to-one mapping.

Four instances were selected from HyFlex to test the compatibility of the transformed FSP in the U/EA standard, as listed in Table 6.3. Even though the instances seemed to be among small-scale instances in HyFlex,

Table 6.3: Four benchmark FSP instances used in the development of the SPOT algorithm

No.	Id	Size ($n \times m$)	% excess the best-known		Resemblance r (%)
			Suboptimum 1	Suboptimum 2	
0	TA081	100×20	1.0158	1.0803	79.76
1	TA082	100×20	0.8895	1.0189	75.50
2	TA083	100×20	0.7176	0.7973	87.72
3	TA084	100×20	0.8614	0.9890	75.33
				Approximability $\bar{r}$ (%)	79.58

they were actually complicated enough[50] to test and to train the SPOT hyper-heuristic. In each instance, two arbitrary suboptima were selected to check the resemblance between them. The approximability $\bar{r}$ was 79.58%, as shown in Table 6.3. Both approximability and generality were greater than those in Chapter 5, respectively. Therefore, the transformed FSP was regarded as compatible to the SPOT hyper-heuristic.

6.3.1.2 Design of decision attributes

Designing the attributes to meet the U/EA^2 standard was the last task in P1. The design involved the definitions of the groups of competitors, the raw attributes and the label(s), similarly to that in Chapter 5.3.1. Each variable of a "following" was considered as a group consisting of two competitor assignments: 0 and 1. Therefore, there were $n^2 - n$ groups and $p = 2$ assignments in each group. The function `getCompetitiveGroups`() returned an array of $2n^2 - 2n$ elements of assignments, with $n^2 - n$ unique identities of groups.

Many existing heuristics, such as Palmer (1965), Campbell et al. (1970)'s CDS, Gupta (1975) and Dannenbring (1977)'s RA, have proposed many

[50] Evidence shows that they are still unsolved.

Table 6.4: The $5m+3$ raw attributes designed for a job J_i against another job J_j in the FSP domain ($i \neq j$)

Raw att	Cols	Definition	Reference
Mean t	1	avg $(t_{i,1}, t_{i,2}, \ldots, t_{i,m})$	
Std dev	1	stdev $(t_{i,1}, t_{i,2}, \ldots, t_{i,m})$	
Min t	1	min $(t_{i,1}, t_{i,2}, \ldots, t_{i,m})$	
Max t	1	max $(t_{i,1}, t_{i,2}, \ldots, t_{i,m})$	
Dist	1	$\sum_{2\leq k\leq m-1}\left[(m-k)\times\lvert t_{i,k}-t_{j,k-1}\rvert\right]$ $+t_{i,1}+t_{i,m}$	
Palmer	1	$\sum_{1\leq k\leq m}[-(m-2k)\times t_{i,k}]$	Palmer (1965)
Gupta	1	$\text{sgn}^{\dagger}(t_{i,m}-t_{i,1})/\min(t_{i,1}+t_{i,2}, t_{i,2}+t_{i,3},$ $\ldots, t_{i,m-1}+t_{i,m})$	Gupta (1975)
Real time	m	$\{t_{i,k}\lvert 1\leq k\leq m\}$	
CDS part1	$m-1$	$\{\sum_{1\leq l\leq k} t_{i,k}\lvert 2\leq k\leq m\}$	Campbell et al. (1970)
CDS part2	$m-1$	$\{\sum_{k\leq l\leq m} t_{i,k}\lvert 1\leq k\leq m-1\}$	Campbell et al. (1970)
RA part1	$m-1$	$\{\sum_{1\leq l\leq k}(m+1-k)t_{i,k}\lvert 1\leq k\leq m-1\}$	Dannenbring (1977)
RA part2	$m-1$	$\{\sum_{k\leq l\leq m}(m+1-k)t_{i,k}\lvert 2\leq k\leq m\}$	Dannenbring (1977)
$\sum$	$5m+3$		

†: the sign function $\text{sgn}(x) = \begin{cases} 1 & x>0, \\ 0 & x=0, \\ -1, & x<0. \end{cases}$

decision attributes. Given a variable $fl(i,j)$[51], there are $ra = 5m+3$ raw attributes defined, as shown in Table 6.4. The first column in Table 6.4 shows the name of a raw attribute. The second column is the number of raw attributes for each category. The third column presents the equations to calculate the attributes. Some references are given in the last column. As a result, the function FSP_UEA2.`getRawAttributes`() returned a 2D array (in $2n^2-2n$ rows and $5m+3$ columns) of Boolean values. The class SPOT then generated $2\times ra$ columns of normalized attributes, because there were only two competitors in each group and some redundant attributes could be removed. A generated data table had $10m+6$ columns and $2n^2-2n$ rows. The data table can be a subset in the U/EA^2 attribute space.

[51] A job J_i against a competitor J_j.

Because the group was defined with the Boolean values of the fl function, the label was set to the fl function and its possible values were 0 and 1. As a result, the objective of finding a permutation was transformed into finding a matrix of "following" relations. Examples of generated data tables of labeled training data and unlabeled test data can be found in Table B.5 and Table B.7 in Appendix B.

6.3.2 P2: Parameter determination

There are two important parameters to be determined one by one in this phase, as described in Section 3.3.1. The first parameter is the size of subproblems n', which is numeric. The other is the method of sampling for generating subproblems, and it is non-numeric.

6.3.2.1 The size n' of subproblems

The tests of n' were set up as follows. The targets of sampling in the FSP domain were the jobs, excluding the machines. The reason was that the number of columns for training was a (linear) function against the number m of machines. If the machines were sampled, the attributes in the training data and the attributes in the test data must be inconsistent. *Random selection* sampling on jobs was employed to determine the size n'. The function SPOT.`run`() was executed without the final step of generating LLHs. The solving time of a subproblem was set to 6 seconds for the PHunter algorithm. The algorithm selection was not enabled for two reasons. The main reason was that the algorithm selection could noticeably reduce the precision of the machine learning. The second reason was the intent to test the performance

(49,48,68,74,42,9,99,67,75,46,4,61,89,58,30,88,18,78,13,97,34,64,71,35,1,69,26,14,32,51,25,33,0,98,31,47,81,8,12,70,52,94,29,83,96,10,2,63,7,92,65,66,86,45,22,59,19,3,90,21,60,95,77,39,79,53,6,82,41,43,72,50,84,57,27,62,37,24,44,91,36,85,73,28,16,76,5,40,15,80,55,23,87,17,11,93,20,38,54,56)

(i) A predicted permutation by the classifiers in the SPOT algorithm

(48,49,75,4,64,30,95,99,97,67,80,26,68,89,42,57,65,54,94,23,31,45,58,74,32,9,71,46,90,72,78,62,34,92,70,13,18,47,15,96,29,60,76,37,25,59,33,14,22,6,77,69,41,83,87,12,61,21,82,51,84,27,36,79,2,7,24,8,66,0,39,35,1,17,93,50,11,88,63,86,55,3,43,73,53,16,91,5,52,44,85,40,10,81,98,19,20,38,28,56)

(ii) An NEH permutation by the NEH heuristic

(49,48,75,4,99,68,67,42,30,64,89,74,97,9,58,46,26,78,71,18,34,32,13,31,94,65,61,95,47,70,14,25,45,69,33,29,92,96,90,51,57,88,12,35,1,83,72,80,59,0,60,22,8,62,2,23,77,7,6,54,21,37,66,41,63,82,15,76,79,98,52,81,86,84,39,27,10,3,36,24,50,87,53,19,43,73,91,17,44,16,93,11,55,85,5,40,28,20,38,56)

(iii) A weighted permutation by adding up the weights in (i) and (ii)

Figure 6.2: Examples of a predicted permutation, an NEH permutation and a weighted permutation for the FSP instance `TA082`

of the supervised learning on a relatively large number (tens to a couple of hundreds) of columns of data. The computation time of the embedded subproblem solver PHunter was also set to six seconds. After training classifiers and predicting labels, the generated matrix was transformed to the weights of summations of "following" back to the given variables for sorting a permutation of jobs. Similar to the weighted distance which combined the general results of learning and variable-specific data of lengths of edges (Xue, Chan, et al., 2011), the weighting in the FSP domain also considered a random permutation by NEH to calculate the weighted summations, with each permutation weighted as a half. An example is shown in Figure 6.2.

The indicator used in the tests was r, which represented the average resemblance between the "following" matrix of a given permutation to those of the two suboptima listed in Table 6.3. The meaning of r could also be regarded as the average *coverage of suboptimum assignments of "following"*. Similarly to the development for the TSP, it is clear that $0 \leq r \leq 1$ and a greater value of r usually stood for a better result of learning and a higher quality of including more promising assignments. It should be noted that the labels generated by classifiers could not be directly used to measure the coverage r before converting to a permutation. There were possibly a lot of inconsistencies in a matrix of predicted Boolean values, such as $fl(i,j) = fl(j,i) = 0$. When the labels were converted to weights and a new "following" matrix was generated, the indicator r can be used to measure the average coverage.

Five values of n' were tested, as shown in Figure 6.3. One hundred independent tests were carried out for each value for each instance. In the "SPOT (direct)" entry, the jobs were sorted by the predicted weights in a descending order. In the "SPOT (weighted permutation)" entry, the jobs were sorted in a combined weight, as shown in Figure 6.2. The "longest-total-time-first" presents a referential r of the permutation in the descending order of the summation $\sum_{1 \leq j \leq m} t_{i,j}$ of the time of a job J_i. The "NEH" shows r of the results of Nawaz et al. (1983)'s NEH heuristic which initialized with the longest-total-time-first sequence. The "average of random NEHs" refers to the average value of 100 NEH heuristics initialized with a randomized permutation.

It can be observed from Figure 6.3 that the indicator r on the results of

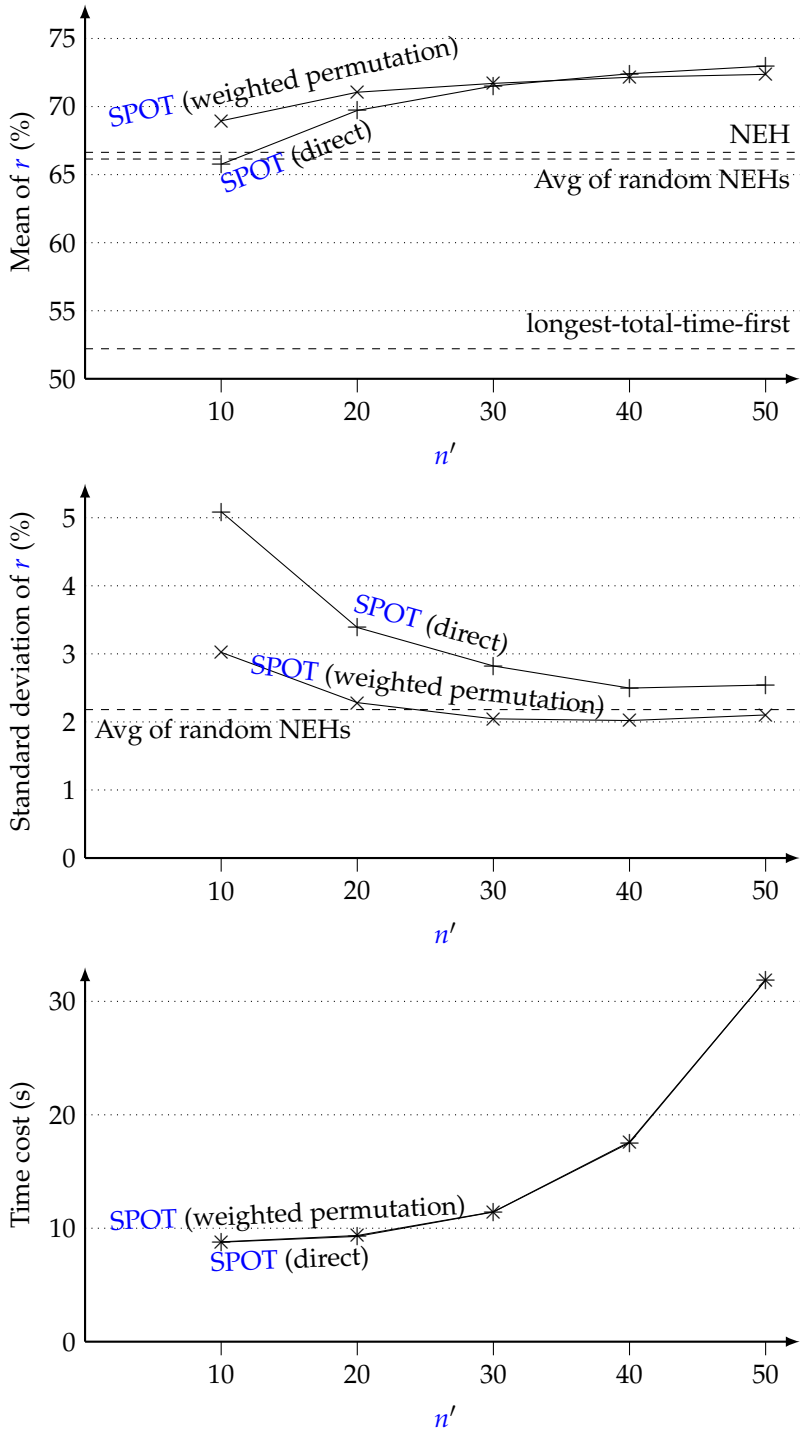


Figure 6.3: Average test results for determining n' in the development of the SPOT algorithm, on the four FSP training instances (100 runs)

the SPOT hyper-heuristic were noticeably greater than for the longest-total-time-first sorting and greater than the results of the NEHs when $n' \geq 20$. When the size n' increased, both r and the time spent on learning gradually and steadily increased. However, the two entries of SPOT methods were too close on both r and time when $n' \geq 30$, therefore, the standard deviation of r is included in Figure 6.3 to check the variances. The size n' was finally set to 30 after a careful balance between r (effect), time cost and, especially, the possibility of training the classifiers from parallel subproblems, like those described in Section 5.3.2.The weighted permutation is selected, because it led to less variance and there was no significant differences on the time cost and the means when $n' = 30$.

6.3.2.2 The methods of sampling for subproblems

Segment selection is another sampling method, which chooses a number of consecutive jobs from a high-qualified permutation. Because all the instances were randomly generated, a random order or a noncompetitive permutation, such as the conventional longest-total-time-first sorting in Figure 6.3, might not generate samples that were much different from those by random selection. The segment selection in this chapter was based on the effective results from the NEH heuristic.

Two groups of tests were carried out on the sampling methods, as shown in Table 6.5 and Table 6.6. In the first group in Table 6.5, the segment selection was compared with the random selection. It was found that the segment selection seemed non-competitive against the random selection. In the other group of tests in Table 6.6, two subproblems were selected in parallel. The

Table 6.5: Average results and significants of r in tests on different subproblems sampled, where $n' = 30$ (100 runs, significant values of p in bold)

Instance	Segment selection (%)	Random selection (%)	ANOVA	
			F	p
0	72.845	72.853	0.00	0.96
1	72.888	73.054	1.43	0.23
2	71.451	72.174	15.29	**0.00**
3	67.726	68.733	34.61	**0.00**
Average	71.227	71.703	—	0.30

Table 6.6: Average results and significants of r in tests on different combination of parallel subproblems sampled, where $n' = 30$ (100 runs, significant values of p in bold)

Instance	Segment + random selections (%)	Random $\times 2$ selections (%)	ANOVA	
			F	p
0	72.569	74.785	77.31	**0.00**
1	74.131	74.962	4.86	**0.03**
2	72.494	73.421	53.46	**0.00**
3	70.563	71.566	68.03	**0.00**
Average	72.439	73.183	—	**0.01**

" Segment + random selections" column in Table 6.6 is the "Random $\times 2$ selections" column and denotes two parallel random selections. Given the methods of sampling as categories, the F values and the significance p are shown in the last two columns in both tables, respectively.

It can be observed from Table 6.5 that the random selection resulted in significant larger values of r in two instances and the average r of the random selection was $+0.476\%$ more. According to Table 6.6, the parallel selections with two random selections were significantly better than those with a segment selection and a random selection in three instances, and the average result was $+0.744\%$ more on r and the improvements were very significant ($p = 0.00$). The final sampling method was set to two

Table 6.7: Average results and significants of time of generation ($time_{overall}$) in tests on different combination of parallel subproblems sampled, where $n' = 30$ (100 runs, significant values of p in bold)

Instance	Segment + random selections (s)	Random ×2 selections (s)	ANOVA	
			F	p
0	25.87	26.12	1.33	0.25
1	25.37	25.59	0.97	0.33
2	25.70	25.71	0.00	0.97
3	25.77	25.59	0.78	0.38
Average	25.68	25.75	—	0.48

parallel random selections. In fact, Kadioglu et al. (2010) also noticed that "tuning on a collection of instances helps prevent over-tuning and allows parameters to generalize to similar instances". The difference between the selected method and the longest-total-time-first in Figure 6.3 on r was a considerable +20.976%. The noticeable difference indicated that some LLHs generated by the SPOT algorithm should be capable of outperforming their base algorithms significantly.

In addition, the time for generating the predicted permutations is shown in Table 6.7. It can be observed that there was no significant difference in any of the instances (as well as the average value) on time of the SPOT hyper-heuristics for the two parallel methods. In summary, the two parallel random selections were able to sample expected subproblems for training the classifiers in about the same time.

6.3.3 P3: The generation of new LLHs

In the following, the labeled values of the following relationship are converted to permutation to make the results more consistent. As designed

and tested, the weighted permutation was used to modify some mutation LLHs provided in HyFlex in the function FSP_UEA2.`setIntLabels`(). The LLHs that were modified, and their modifications are listed in Table 6.8. The basic idea was in changing a mutation heuristic to a guided mutation and changing a local search to a guided local search. The two LLHs in the ruin-and-recreate category are also considered as multi-variable mutations. No. 9 and No. 10 LLHs were modified to *greedy random* local search LLHs by a greedy initialization and a greedy randomization. A rejection function was employed in the greedy randomization for modifying the search sequences in No. 9 and No. 10 LLHs. Given a swap of two jobs (i, j), there was probability to reject the swap:

$$1 - \frac{\min(|predicted(i) - location(i)|, |predicted(j) - location(j)|)}{n}. \quad (6.5)$$

The rejection function denoted that the jobs closer to the predicted locations rejected to be shuffled in a higher probability. No. 2 LLH was a complete randomization for all the jobs. No. 3 LLH was a complete randomization of a part of the permutation. No. 4 LLH was a NEH using the input permutation as the initial sequence. They were not included in the modification. The local search heuristics No. 7 and No.8 were not included in the modification. The main reason was the scopes of the two local search LLHs were not modified by the new permutations, Section 6.5 gives more details. Similarly to those LLHs in the TSP domain, six new LLHs were *generated* as Nos. 0′, 1′, 5′, 6′, 9′ and 10′. The original LLHs are called the base algorithms.

Table 6.8: The modified LLHs for FSPs and modifications by the SPOT heuristic generation

Type	Id	Original behavior	Modified behavior
Mutation	0	Reinserting a random job to a random index.	Choosing $3^{\dagger}$ random indices, doing the most favorite reinsertion (1 in 6).
	1	Swapping two random jobs.	Choosing $3^{\dagger}$ random indices, doing the most favorite swap (1 in 3).
Ruin-and-recreate	5	Removing some random indices and inserting back by calling iterated NEHs.	On each removal, choosing $3^{\dagger}$ random indices, always removing the least favorite index.
	6	Removing some random indices and inserting back by calling stepper NEHs.	Same as above.
Local search	9	Randomizing a sequence, trying width-first steepest descent 1-interchange one by one.	Building a greedy sequence in the descending order of the distant to the predicted index, randomizing the sequence by swaps with a *rejection function*.
	10	Randomizing a sequence, trying width-first first improvement 1-interchange one by one.	Same as above.

†: Effective parameters according to tests, not rigorously examined in thesis.

6.4 Experiments and Observations

This section describes two groups of tests. In the first group of experiments, the individual performance of each new LLHs was examined, with comparisons to their base algorithms. In the other group of tests, the five new LLHs replaced their base algorithms. The heuristic selection method PHunter solved the updated domain consisting of new LLHs. The results were compared to the solving of PHunter with the original LLHs and the winners in the FSP domain in CHeSC 2011.

All the tests were carried out on the same instances which had been selected for the CHeSC 2011 competition, i.e. the numbers 1, 3, 8, 10 and 11

shown in Table 6.1. The size n' of the subproblems was set to 30 and two subproblems were generated in two parallel random selections to provide training data, as described in Section 6.3.2. The overall computation time was set to the same as the normalized time in CHeSC 2011, which was equivalent to 10 CPU minutes for a Pentium 4 3.0 GHz computer. All the time costs of the processes in the SPOT hyper-heuristic, such as sampling subproblems, solving the subproblems, learning instance-specific knowledge and modifying LLHs, were fully included in the overall time in order to make the results comparable. For the embedded subproblem solver PHunter, the time limit for solving each subproblem was 1% of the overall time, i.e. equivalent to 6 CPU seconds on a Pentium 4 3.0 GHz machine. The time cost of rest processes the in SPOT was not directly restricted.

6.4.1 On the individual LLHs

Table 6.9 shows the results on the final *makespan* of the newly generated LLHs in terms of percentage excess over the best-known solutions. There were 100 runs for each configuration and each problem instance. In each run, a random initial tour was generated by the NEH with a random seed. All the LLHs started from the same initial tour independently. One-way ANOVA tests were also included to check the significance of the changes. The third column in Table 6.9 is the test instance, and the fourth and the fifth columns show the excesses of the best and the average makespans found by the base algorithms in HyFlex, respectively. The sixth and the seventh columns indicate the best and the average makespans found by the new LLHs, respectively. The eighth column denotes the change in the average

excess, where improvements (negative numbers) are written in bold. The last two columns show the F values and the confidence p examined by the one-way ANOVA tests. The significant values ($p \leq 0.05$) are in bold.

It can be observed from the eighth column that all the newly generated LLHs in the ruin-and-recreate and local search classes generally returned better (negative in % excess) permutations than their base algorithms. The average improvements in the ruin-and-recreate class were statistically significant and was more than those in the local search class. The differences between the new LLHs and their base algorithms in the local search class were *almost* significant, though the null hypophyses could not be fully objected ($p = 0.10, 0.08$). In comparison, the new LLHs in the mutation class were neither significant nor robust. The observations on the LLHs in the latter two classes were consistent with the interpretation and prediction according to the indicator r in Section 6.3.2. The insignificant and unstable improvements in the mutation category were because of the main objective of mutation operations, which aim at diversifying a given solution.

The *time cost* of *generating* the LLHs is shown in Table 5.7. The third to the sixth columns show the minimum, the average, the maximum values and the standard deviation of the time cost of the whole SPOT hyper-heuristic for each instance. It should be noted that a limit of 12 equivalent CPU seconds for solving two subproblems was included in the data. It can be observed that the maximum time cost in the large-scale instance with 500 jobs and 20 machines was about 60 seconds. Therefore, the SPOT could carry out instance-specific heuristic generation in an "on-the-fly" fashion for practical problems, for example, in a 10 minute time.

Table 6.9: Test results on the makespan of the generated LLHs and their base algorithms (10 runs, 100 iterations in each run, Depth of search = 1, Intensity of mutation = 1, improvements on average performance in bold, significant values of p in bold)

Category	Group	Inst	% excess by the base LLH		% excess by the new LLH		Δ avg	ANOVA	
			Min	Avg	Min	Avg		F	p
Mutation	No. 0	1	4.642	7.004	4.432	7.053	0.049	1.41	0.23
	versus	3	4.100	6.435	4.291	6.378	**-0.057**	1.93	0.17
	No. 0′	8	2.342	3.069	2.206	3.081	0.012	1.00	0.32
		10	3.734	5.488	3.698	5.484	**-0.004**	0.02	0.90
		11	2.154	2.827	2.063	2.837	0.010	0.71	0.40
				Avg improvement (%)			-0.096	Avg	0.40
	No. 1	1	4.351	7.485	4.561	7.447	**-0.038**	0.59	0.44
	versus	3	4.371	6.865	3.876	6.834	**-0.031**	0.40	0.52
	No. 1′	8	2.247	3.175	2.244	3.167	**-0.008**	0.41	0.52
		10	3.975	5.760	3.957	5.770	0.009	0.05	0.83
		11	2.154	2.919	2.108	2.924	0.005	0.15	0.70
				Avg improvement (%)			**0.177**	Avg	0.60
Ruin-and-recreate	No. 5	1	4.739	6.510	4.221	6.188	**-0.321**	115.81	**0.00**
	versus	3	3.892	5.742	2.951	5.513	**-0.230**	61.40	**0.00**
	No. 5′	8	2.391	3.103	2.195	3.012	**-0.092**	65.62	**0.00**
		10	3.805	5.415	3.868	5.381	**-0.033**	1.58	0.21
		11	1.843	2.827	2.097	2.766	**-0.061**	31.23	**0.00**
				Avg improvement (%)			**2.931**	Avg	**0.04**
	No. 6	1	3.396	5.127	3.105	4.867	**-0.260**	95.50	**0.00**
	versus	3	2.983	4.633	2.616	4.468	**-0.165**	37.20	**0.00**
	No. 6′	8	1.836	2.510	1.791	2.426	**-0.084**	70.67	**0.00**
		10	3.037	4.451	2.814	4.390	**-0.061**	6.18	**0.01**
		11	1.710	2.283	1.627	2.225	**-0.057**	36.93	**0.00**
				Avg improvement (%)			**3.171**	Avg	**0.00**
Local search	No. 9	1	3.510	5.097	3.299	5.016	**-0.081**	10.24	**0.00**
	versus	3	2.999	4.565	2.903	4.507	**-0.058**	5.20	**0.02**
	No. 9′	8	1.821	2.424	1.761	2.399	**-0.024**	5.71	**0.02**
		10	2.930	4.240	2.975	4.223	**-0.018**	0.75	0.39
		11	1.638	2.165	1.551	2.147	**-0.018**	3.72	**0.05**
				Avg improvement (%)			**1.023**	Avg	0.10
	No. 10	1	3.574	5.170	3.607	5.130	**-0.041**	2.65	0.10
	versus	3	2.903	4.621	3.063	4.577	**-0.044**	3.23	0.07
	No. 10′	8	1.829	2.441	1.701	2.421	**-0.020**	3.98	**0.05**
		10	2.966	4.300	2.930	4.267	**-0.033**	2.51	0.11
		11	1.566	2.185	1.612	2.166	**-0.019**	4.00	**0.05**
				Avg improvement (%)			**0.839**	Avg	0.08

Table 6.10: Average overall time ($time_{overall}$) of generating new LLHs (10 runs)

Instance	Size	Time (s)			
	($n \times m$)	Min	Avg	Max	Std dev
1	100×20	11.19	13.51	16.77	1.61
3	100×20	12.11	14.26	17.24	1.59
8	500×20	59.25	62.77	65.38	2.06
10	200×20	19.35	21.65	24.87	1.87
11	500×20	60.19	62.41	64.80	1.50

Table 6.11 shows the computation time of the new LLHs and their base algorithms for solving problems. No. 1 and No. 2 LLHs only changed the indices of some jobs without any iteration or backtracking. Therefore, the computation time and the differences in computation time in the first two groups was not noticeable. The time was also hardly noticeable in the tests for instances No. 1 and No. 3. Slight increments on time could be seen in the results of the LLHs No. 5′ and No. 6′, while slight reductions could be observed in those of the LLHs No. 9′ and No. 10′. Both the percentages of the increments and the reductions were no more than 5%. In summary, the newly generated LLHs spent almost the same amount of time as their base algorithms, respectively. The new LLHs are eligible to replace their base algorithms in HyFlex in practice.

Figure 6.4 represents the non-dominance of four new LLHs. The tests were carried out on all the five 100×20 instances implemented in HyFlex. It can be observed from Figure 6.4 that the two LLHs in the ruin-and-recreate category were considerably improved in solution quality and the two in the local search class were slightly improved in solution quality. All the differences on time were very limited. The relevant significances are included in Table C.6 in Appendix C. In summary, it is confident to have four new and

Table 6.11: Average run time of each LLHs (10 runs, 100 iterations in each run, instrument error < 0.16 ms)

Inst	Average run time of LLHs in the four group (ms)																	
	0	0′	Δ	1	1′	Δ	5	5′	Δ	6	6′	Δ	9	9′	Δ	10	10′	Δ
1†	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0
3†	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0
8	0	0	0	0	0	0	40.7	40.7	0.0	13089.5	13051.7	-37.8	120.8	121.0	0.2	120.9	120.9	0.0
10	0	0	0	0	0	0	6.7	6.6	-0.1	831.9	831.3	-0.6	19.1	19.3	0.2	19.0	19.1	0.1
11	0	0	0	0.1	0	0	40.8	40.8	0.0	13067.3	12947.3	-120.0	120.9	121.1	0.2	120.7	120.4	-0.4
Avg imp (%)									0.3			0.4			-0.5			-0.1

†: Not included in the average improvements due to the level of instrument error.

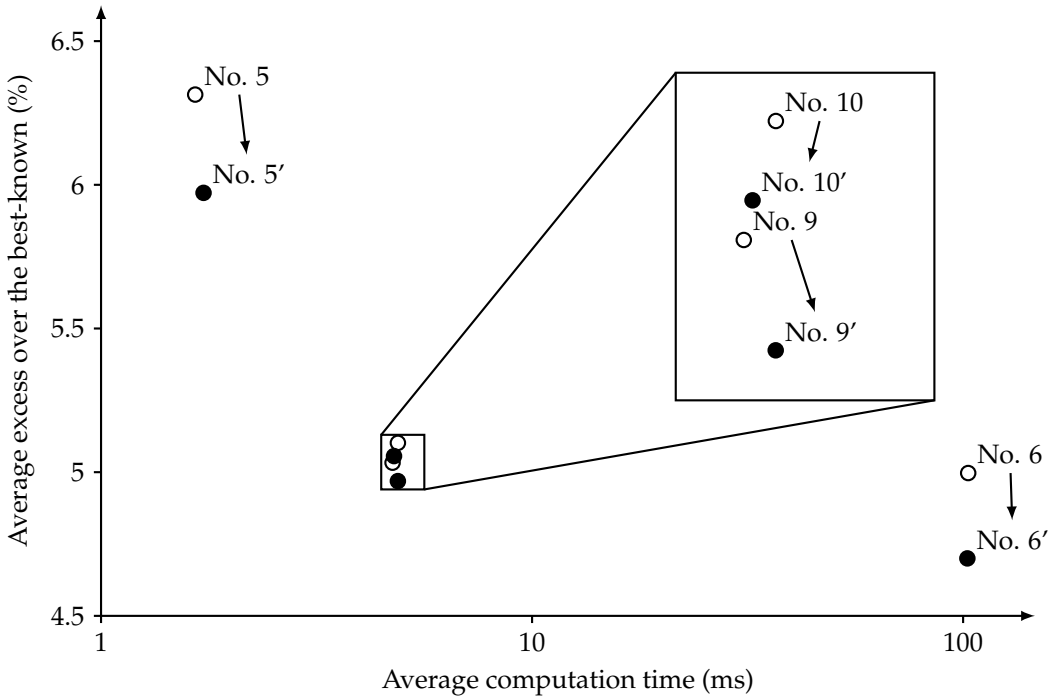


Figure 6.4: Comparisons between the new LLHs and their base algorithms for solving the 100×20 FSP instances `TA81`, `TA82`, `TA83`, `TA84 and TA85`

non-dominated LLHs in the ruin-and-recreate and the local search categories in the results of the SPOT heuristic generation.

Table 6.12: The top three hyper-heuristics in the FSP domain in CHeSC 2011

No	Algorithm	Score	Author
1	ML	39	Mathieu Larose (Université de Montréa)
2	AdaptHH	37	Mısır et al. (2012)
3	VNS-TW	34	Hsiao, Chiang, and Fu (2011)

6.4.2 Comparisons with other hyper-heuristics

Based on the four new LLHs generated by the SPOT algorithm, many existing hyper-heuristics, especially heuristic selection methods, can benefit. A typical and straightforward way is to replace the base algorithms by the generated LLHs. After replacement of their base algorithms, the set of fifteen LLHs of the FSP domain in HyFlex consisted of the six new LLHs. The heuristic selection method PHunter was also employed to run on the basis of an updated LLH set. The combination of the SPOT preprocessor and the body of the PHunter is noted as the SPOT-PHunter in the following tests.

The HyFlex framework and the competition CHeSC 2011 also provided reliable results for twenty hyper-heuristics in the FSP domain[52]. The top three hyper-heuristics are listed in Table 6.12. There were five instances selected, as shown in Table 6.1, so a maximum possible score was 50 for each entry.

The SPOT-PHunter was tested as a normal algorithm player in CHeSC 2011. The parameters of the SPOT part were the same as determined in this chapter, and the parameters of the PHunter part were the same as determined in Chan et al. (2012). All the six new LLHs were used to replace their base algorithms. The tests were conducted for the five instances selected in CHeSC

[52]See http://www.asap.cs.nott.ac.uk/external/chesc2011/raw-results.html.

Table 6.13: Median makespan of the PHunter, the SPOT-PHunter and three other hyper-heuristics in the FSP domain in HyFlex (31 runs, best results in bold for each instance)

Inst	Best results† in CHeSC 2011			Test results in thesis		
	ML	AdaptHH	VNS-TW	PHunter_{64bit}	SPOT-PHunter	ANOVA p
1	6245	**6240**	6251	6246	6242	0.55
3	**6323**	6326	6328	6350	**6323**	**0.00**
8	26800	26814	26803	26822	**26790**	**0.00**
10	11384	**11359**	11376	11375	**11359**	**0.01**
11	26610	26643	**26602**	26609	26604	0.28

†: Results on a 32-bit Java.

2011, and each was tested 31 times. The time cost in each test was equivalent to 10 minutes on a Pentium 4 3.0 GHz CPU. The median values were used for scoring, as shown in Table 6.13. The second to the fourth columns show the median solutions found by the top three hyper-heuristic approaches in CHeSC 2011. The fifth and the sixth columns show the median results of the PHunter and the SPOT-PHunter. The last column shows the significance of one-way ANOVA test. The best results for each instance are written in bold. Two entries of the PHunter_{64bit} and the SPOT-PHunter were added to the set of twenty hyper-heuristics. The updated scores, according to the scoring system, of the top five hyper-heuristics are shown in Figure 6.5. It should be noted that the tests in CHeSC 2011 were carried out on a 32-bit Java environment.

It can be seen from Table 6.13 that the SPOT-PHunter found the three best results in the five instances. The final score of the SPOT-PHunter was 43 out of 50, as shown in Figure 6.5. The runner-up was the AdaptHH with a score of 31. The PHunter_{64bit} solved instances with base algorithms instead of newly generated LLHs. The PHunter_{64bit} gained a score of 20.5,

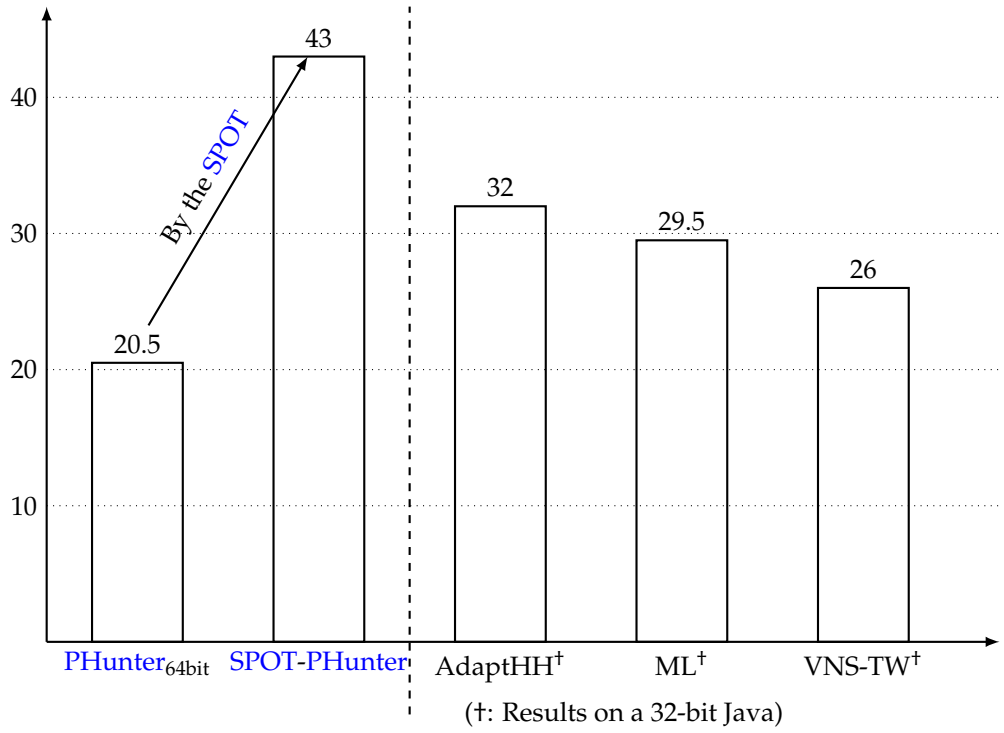


Figure 6.5: Scores of the PHunter, the SPOT-PHunter and three other hyper-heuristics in the FSP domain in HyFlex

which was the lowest in the five hyper-heuristics. The SPOT-PHunter won the $\text{PHunter}_{\text{64bit}}$ in every instance according to Table 6.13. Furthermore, the difference between the result of the $\text{PHunter}_{\text{64bit}}$ and that of the SPOT-PHunter was significant in three instances out of the five, as shown in Table C.7 in Appendix C. The observations on the scores and the improvements meant that the SPOT heuristic generation *generally improved* the heuristic selection of the $\text{PHunter}_{\text{64bit}}$ in solving in every FSP instance, where the improvements were significant in three instances.

The main reason for the significant improvements is the observations and interpretation of the significantly non-dominated LLHs, as presented in Section 6.4.1. Another reason was No. 7 and No. 8 local search LLHs did not

Table 6.14: A flawed transformation of FSPs that resulting in a very low approximability

No.	Id ($n \times m$)	% excess the best-known		Resemblance r (%)
		Suboptimum 1	Suboptimum 2	
0	TA081	1.0158	1.0803	7.00
1	TA082	0.8895	1.0189	2.00
2	TA083	0.7176	0.7973	18.00
3	TA084	0.8614	0.9890	6.00
			Approximability $\bar{r}$ (%)	8.33

"homogenize" the differences brought by the new LLHs. That was probably because the depth of search[53] was only 1 in those local search LLHs. If a 3-job interchange local search or a 4-job interchange was implemented, the new LLHs might not return significantly different results.

6.5 Discussion

The transformations in this chapter and Chapter 5 met the requirements of the U/EA standard and had acceptable approximability, although, there were some flawed transformations that failed to do so. For example, given an FSP problem, let the position (index) of each job in the permutation be a variable. The transformed problem seemed to meet the standard of U/EA, however, the approximability was very low, as shown in Table 6.14. The approximability of the transformed model was 8.33% which was too low for the SPOT algorithm.

The reason why the main model in this chapter was converting the relationship of "following" into the variable, is based on the following observa-

[53] It means the true integer depth of local search (range: $1, 2, \ldots, n$), not the decimal "Depth of search" parameter (range: 0 to 1) in HyFlex.

tion. Taking the two suboptima for the instance No. 0 as examples, as shown in Table A.1 in Appendix A, one can easily find that both permutations end with the substring "$\{\ldots, 48, 12, 6, 40\}$". The first suboptimal permutation started with a substring "$\{53, 77, 58, 21, \ldots\}$", and the other started with a substring "$\{53, 21, 46, 0, \ldots\}$". In the four extreme positions in the heads and the four in the tails, the two suboptima had six of the same jobs and five of them were exactly in the same positions. After further investigation, it was not difficult to discover that the relationship of "following" was much more meaningful and robust for suboptima.

It could be concluded that the parameter n' is more important than the method of sampling, according to the values of resemblance r in Figure 6.3, Table 6.5 and Table 6.6. It is clear that the differences of r varied much more in Figure 6.3 than in the other two tables. The observation, in fact, verified the development process of the determination n' first, with the method set to random selection.

The LLHs generated by the SPOT in this chapter were mutations and some local search heuristics with a slight changing in their behavior. No. 7 and No. 8 local search LLHs did not benefit from the generated permutations. The main reason was that the depths of search of the four local search LLHs were, in fact, equal to 1. Both were variants of the 1-interchange local search. The differences were mostly on the criterion of acceptance (steepest descent or first improvement). The scopes of search of the two local search methods did not change noticeably by reordering their search routes according to a generated permutation. The capability of the 1-interchange local search actually was also limited. As a result, the differences on the makespan by

the newly generated LLHs, as shown in Table 6.9, seemed passed onto the results of the SPOT-PHunter, as shown in Table 6.13. In fact, the predicted permutation by the SPOT hyper-heuristic could possibly benefit the local search LLHs by extending the scope of search to a higher level. For example, if a 1-interchange was very favorite according to learning results but failed to improve the permutation, a conditional 2-interchange could be enabled.

Chapter 7

Discussion and Conclusions

> In the confrontation between the stream and the rock, the stream always wins, not through strength but by perseverance.
>
> Brown Jr., H. Jackson

Heuristic generation is a subclass of hyper-heuristics. Instead of solving a given problem directly, a heuristic generation approach *generates new* heuristics. This thesis presents a heuristic generation methodology called SPOT (Suboptimum- and Proportion-based On-the-fly Training). The proposed method was implemented and verified in two well-known NP-Complete combinatorial optimization problem domains. Section 7.1 gives some discussion on the findings, implications and limitations while concluding remarks are given in Section 7.2.

7.1 Discussion on the SPOT Methodology and Findings

In summary, the SPOT hyper-heuristic is a novel methodology aiming at improving existing heuristics and data structures with instance-specific information. Furthermore, there are new standard and new indicators developed for SPOT so that the instance-specific information can be obtained from optima, suboptima or suboptima of subproblems for a given problem. The limited scale of learning enables the capability of on-the-fly execution of SPOT. Applications in the well-known NP-Complete problem domains such as the TSP domain and the FSP domain were successful and verified the SPOT hyper-heuristic generation in general.

7.1.1 Principal findings

Referring to the development procedures and the experimental results, the presented SPOT hyper-heuristic was shown to be capable of conducting an on-the-fly supervised learning for finding instance-specific information and generating new heuristics according to the obtained instance-specific information. Particularly, it was proven to generate significantly non-dominated new heuristics for *large-scale* FSP benchmark instances *in a few minutes*. In terms of operations, SPOT contains three phases: (i) input to the problem with certain regulations, (ii) suboptimum- and proportion-based supervised learning for the instance-specific information and (iii) modify (or create) of heuristics.

The input problems regulations included the U/EA standard and the indicator of "*compatibility*". As a requirement of SPOT, the U/EA standard

of the input combinatorial optimization problem was applied in the two domains. Three transformations of combinatorial optimization problems, one for the TSP and two for the FSP, were designed to remove the inter-variable hard constraints so that a transformed problem can meet the U/EA standard. An indicator of "resemblance" r was employed to measure the compatibility of each transformation. The *generality* of applying SPOT to transformed problems was measured by the similarities between the average resemblance values of the instances. The *approximability* of transformed problems was measured by the overall average resemblance value of all the instances. Later tests and discussion verified the U/EA standard and the compatibility. According to experience, some problem domains are natural for the U/EA standard and the compatibility, such as the TSP domain. Some others with strong inter-variable hard constraints or strong inter-variable correlations are hard to meet. An example is the personnel scheduling problem involving complex contracts and licenses (staff skills) and a 24-hour job demand.

A distinct feature of the SPOT hyper-heuristic is the capability of learning approximate information on the basis of *suboptima* and sampled *proportions* instead of optima and the original problem. This is the key to the success of SPOT. As stated in the Chapter 3, the capability came from the U/EA standard and the compatibility of problems. Attributes were developed to represent information from the suboptima of subproblems. In this thesis, an ordinary measurement was considered as a "raw attribute". An automated attribute generation procedure helps populating many "normalized attributes" and assists the data in the learning process to meet the U/EA[2]

standard. Many attributes were about comparisons and increments, which have no unit but represent certain concerns of competition in the possible assignments. The instance-specific information about suboptimality was in the form of representing competition in the opponent assignments. In fact, besides the stochastic machine learning, there could also be many other candidates for explicit or implicit learning, such as the backbone (W. Zhang & Looks, 2005) and tour-merging (Cook & Seymour, 2003) in the TSP domain.

The generation of *new* heuristics, in fact, would be much more often in the form of *modifying* existing heuristics rather than creating new ones completely. The key process in generating new heuristics is the transformation from instance-specific information to numeric or non-numeric objects that directly interacted with the search heuristics. This thesis presents some examples on how to utilize the obtained instance-specific information to modify (or guide) the search in existing heuristics. For example, in FSP, the predicted permutation based on the "following" relationship can advise the promising mutation operation from a number of random mutation operations. This representation (or interpretation) might not be necessary if the learning involved was not stochastic learning, such as the backbone (W. Zhang & Looks, 2005) in the TSP domain.

In the practical development of SPOT, there were two main parameters to be determined in the SPOT heuristic generation. One was the size of the subproblems, and the other was the method(s) to sample subproblems. The first one was more important than the second one, according to the U/EA standard. The indicators, one or more, represent the resemblance r and help identify better configurations. Other numeric and non-numeric

parameters are considered as secondary and adopted from the literature. These parameters may, somehow, affect the whole algorithm, though the suggested values (or methods) in the literature would be of low-risk and be more robust in general.

The main *theoretical* findings were the formal definitions on hyper-heuristics, heuristic section and heuristic generation. As far as is known, the definitions are the first exact formulations, especially for the latter two. In brief, if a hyper-heuristic can only return finite or countably infinite different heuristics, it is a heuristic selection approach; if a hyper-heuristic can return some heuristics in uncountable possible algorithms, it is a heuristic generation. The presented formal definitions can distinguish heuristic selection approaches and heuristic selection methods precisely, and most of the results are consistent with previous work. Furthermore, the scopes of the notions were considerably extended. For example, the backbone (W. Zhang & Looks, 2005) and the tour-merging (Cook & Seymour, 2003) in the TSP domain could be regarded as heuristic generation methods by the definition.

7.1.2 Interpretation and implications of findings

First, all the findings *verified* the feasibility, the effectiveness and the efficiency of the SPOT heuristic generation. In fact, the verification did not only come from positive supporting results, such as the significantly improved ruin-and-recreate heuristics in the FSP domain, but also from the neutral and even negative results in terms of solution quality or applicability, such as the insignificant improvements in the TSP domain and the flawed transformation of FSPs. All the positive, neutral and negative cases supported the

values and the prediction of the indicator(s) of the SPOT development.

The two applications of the SPOT methodology in the two NP-Complete problem domains showed that the presented algorithm was capable for *cross-domain* applications. Particularly, every problem domain should be transformed to meet the U/EA standard for a possible effective adaptation of the SPOT hyper-heuristic. The capability of the SPOT would, in fact, depend more on the capability of the developer in regard to transformation than the specific given problem domain or the particular set of given problem instances. It could probably also give a possible solution to algorithms, which are encountering NFL theories. The proposed algorithm aimed at being able to handle cross-domain problems in certain conditions, such as the U/EA standard. However, the U/EA standard is closely related to perturbative meta-heuristics, as mentioned in Section 3.3.1. The SPOT, therefore, could not become a universal solver.

Computer power has been increasing continuously. However, the complexity, such as incomputability and NP-hardness, limited practical polynomial time programs in problem solving. It is very common to see an up-to-date effective meta-heuristic or exact algorithm spending a long time, up to weeks and months, on a large-scale combinatorial optimization problem. Therefore, it is not easy to find optima to discover instance-specific information. The capability of learning instance-specific information from suboptima of subproblems was is most interesting, and is theoretically enabled by the U/EA standard and a high-level of compatibility. The two regulations made the learning and generation feasible and on-the-fly.

In a "compatible" domain, perturbative heuristics such as local search

should be relatively more effective than constructive heuristics. Hence the two applications also mainly focused on perturbative heuristics. New LLHs were generated by modifying existing LLHs. Therefore, it seems the SPOT applications in thesis should be called heuristic "modification" instead of generation. Nevertheless, new constructive heuristics can also be generated. For example, the generating permutations by "direct" weighting in Section 6.3.2 was actually a constructive solution — though it was easily dominated. It would also be interesting to investigate whether the instance-specific information can be applied in exact algorithms. Xue, Chan, et al. (2011) gave a preliminary trial on this topic.

The SPOT heuristic generation presented in this thesis was based on stochastic supervised machine learning. However, it would probably be possible to extract instance-specific information through non-stochastic machine learning methods, such as the backbone (W. Zhang & Looks, 2005) and tour-merging (Cook & Seymour, 2003) in the TSP domain. It is because the essence of the learning phase is to distinguish "promising" assignments from a group of competitors. For the applications without sampling for subproblems, the intersection or the union of assignments from different suboptima, such as the backbone and tour-merging, could provide strong information about suboptima assignments, too. Therefore, it would be interesting to investigate a non-stochastic learning version which continues to use the "sample-learn-generate" framework of the SPOT algorithm on-the-fly.

7.1.3 Interpretation in the context of the literature

Generally, the findings in this thesis were consistent with previous research described in the literature. For example, the application in the TSP domain re-implemented the sampling and heuristic modifications of Xue, Chan, et al. (2011) and partially replicated the decision attributes. The development and results found some trends that were the same as those described in Xue, Chan, et al. (2011). One main difference was that the average coverage was noticeably improved in Xue, Chan, et al. (2011) when it was slightly and insignificantly improved, as described in Chapter 5. Explanations are given in Section 5.5. Another example was the redefinition of the three notions of hyper-heuristics. Although the three definitions in this thesis changed all the notions by increasing their extensions, their intensions remained. As a result, most of the clearly categorized hyper-heuristics, such as those in Burke, Hyde, Kendall, Ochoa, Özcan, and Woodward (2010), could be classified to the same category according to the new definitions.

The sampling for subproblems is the first main step on the SPOT hyper-heuristic, if the transformations are considered as preparations. In fact, the sampling in this thesis was more similar to the sampling in stochastic learning, particularly the Naïve Bayes, and some meta-heuristics, especially the EDAs. There were normal distributions and the kernel method for a Naïve Bayes classifier to use (John & Langley, 1995). There were standard probability-based (Mühlenbein & Paaß, 1996; Mühlenbein, Bendisch, & Voigt, 1996) and factorized distribution-based (Mühlenbein & Mahnig, 1999) estimations in EDAs, too. In this research, both random selection and some

non-random selections were also designed and tested. In some domains, such as the TSP domain, it was better to use subproblems generated by different methods of sampling.

The final results of the experiments showed that the SPOT methodology was very competitive against the winners of CHeSC 2011 on the HyFlex framework. However, it should be noted that the opponents' results were tested on a 32-bit Java, and the results in this thesis were tested on a 64-bit Java. Although there was a performance factor measured by an official program to covert the local CPU time to a Pentium IV 3.0 GHz, the conversion seemed not so precise when a lot of floating computations were involved. One proof was that the PHunter got a slight but steady performance improvement on every single test instance in the TSP domain. Similar observation can also be obtained in the FSP domain, see Table C.8 in Appendix C. However, due to the demanded memory in machine learning, a 64-bit Java had to be chosen for conducting all tests.

7.1.4 Limitations

The formal definitions contributed a few formulations of some important concepts in hyper-heuristics. However, the definitions were, in fact, no more than a few manipulations, which tied the distinguishability of the hyper-heuristic notions to well-known concepts. In addition, the distinguishability of heuristic generation, for example, was only applicable when there was an *infinitely large* problem. Therefore, the given definitions are preliminary and would benefit from modifications and refinements in future.

In this thesis, the methodology of examining the performance of a new

algorithm was based on experiments on benchmark instances and in comparison with well-known effective algorithms in an (hopefully) equivalent environment. There were a number of factors that might cause some errors in the final results of the two domains of applications, as discusses as follows.

Number of test instances: Five test instances, as many as tests in CHeSC 2011, were used in each domain. If there were more instances, the results should become theoretically slightly less biased, if any.

Source of test instances: Although the test instances were well-known benchmark instances, more application domains would avoid biased results, if any. For example, the instance selection in the TSP domain seemed better than that in the FSP domain.

Unstable solutions: The stop criterion was a time limit, so the operations or flops done by a CPU were not stable. A solution was changing the stop criterion to iterations or operations.

Instrument error in measuring time in Java: In the implementations in this study, the time cost was measured by a class ThreadMXBean provided by Java. However, the precision was not very satisfactory. In certain cases, the resolution on time provided by the class was about 160 milliseconds. That was too much in comparison with fast heuristics of one millisecond.

32/64-bit environment of Java: It should be appreciated that there was an official program in CHeSC 2011 to measure a CPU factor for the participant's machine. However, the measurement seemed imbalanced according to the test results in Chapter 5 and Chapter 6. A probable

reason was the float/integer ratio considered in the program might vary in different applications of domains.

CPU and memory performance adjustment: A state-of-the-art CPU and memory are often capable of adjusting their voltages, clocks and performances to save power. Some models of CPUs also mapped one hard core into two virtual cores. This would be an issue in measuring the performance of an algorithm.

LLH implementation: The quality of LLHs also affected the results. If there was some unnoticed error in the LLHs in HyFlex, there would be inherent errors.

One answer to resolve the random errors, such as the unstable solutions and instrument error, was to use the statistical techniques. For example, the one-way ANOVA tests were generally employed in this thesis to compare the differences in average values between two sets of values. Besides the possible sources of errors listed above, the limited sources and scale of benchmark instances in HyFlex and the Formula 1-like scoring system in CHeSC 2011 might also lower the precision and the confidence of the experimental results. Nevertheless, Xue, Chan, et al. (2011)'s results of a similar heuristic generation method showed a good scalability for Euclidean TSP instances, which covered three different sources and from 3,000 to 1,000,000 cities.

There were also some difficulties in the development phase of the SPOT heuristic generation, as follows.

Transformation for the U/EA standard: Although it is not difficult to find a U/EA transformation, transformations cannot be easily borrowed

from one domain to another. For example, in the personnel scheduling domain in HyFlex, some perturbative techniques worked very well. But both transformations in the two applications failed in practice. Constructing a new high-level of compatible transformation may spend a lot of time.

Memory usage of stochastic learning: The demand for memory could be billions of bytes, when there are millions of rows of data records. This problem is particularly serious when there is no attribute selection.

Interpretation of instance-specific information into heuristics: The instance-specific information obtained in this research was in the form of stochastic classifiers. The method of conversion of the classifiers would be important for the final generated heuristics. It is a relative difficult task to find a highly satisfactory conversion.

Although the main framework of the SPOT methodology is presented in thesis, there are a few issues not mentioned:

Parameter analysis of the size of initial raw attributes: In thesis, the size, as a parameter of attribute selection method, was referred to some default value. Further works can be done in examining the size of raw attributes in some given domains.

Parameter analysis of the size of sampled subproblems: In the two applications, the sizes were set to two, which was the minimal number fulfilling the requirements of Definition 12. Further analysis can show the difference of the measurements of compatibility by different sizes.

Descriptive compatibility: One issue is to determine a descriptive scale to interpret the two numbers "generality" and "approximability" in compatibility, e.g., "good" for 0.8 to 1.0 generality and "incompatible" for 0 to 0.05 generality.

Transformation between domains: In the literature, Ruiz-Vanoye et al. (2011) summarized a transformation map between hundreds of NP-Complete problems. Maybe in the future an approximability tree or table can be organized, so that a given domain A can take advantage of a well-defined transformation in another known domain B by mapping itself to B first. A given practical domain A can also take advantage of another well-tackled domain B by mapping, too.

7.2 Conclusions

This thesis presents the SPOT heuristic generation approach for solving combinatorial optimization problems. With the proposed "sample-learn-generate" framework, SPOT samples small-scale *subproblems*, obtains *instance-specific* information from the *suboptima* of the subproblems by machine learning, then generates new heuristics by modifying existing heuristics and data structures accordingly. All the operations are conducted on-the-fly, and it is capable of generating non-dominated new heuristics for large-scale instances very efficiently, such as in the FSP domain. In the development of SPOT algorithm, two standards were incorporated to regulate the problem input and the machine learning data. Besides, an automated attribute generation procedure was introduced for populating more "normalized attributes" for

learning. An indicator of "resemblance" of suboptima is also introduced to measure the *compatibility* of the input problem. Generally, if a problem domain meets the standard of input and its compatibility is high, the SPOT algorithm is expected to be a competitive method in the domain.

Additionally, formal definitions of the hyper-heuristic and its subclasses, heuristic selection and heuristic generation, were proposed in this study. Based on the new definitions, SPOT is a heuristic generation algorithm. These definitions are mostly consistent with existing taxonomies on hyper-heuristics, such as in Burke, Hyde, Kendall, Ochoa, Özcan, and Woodward (2010).

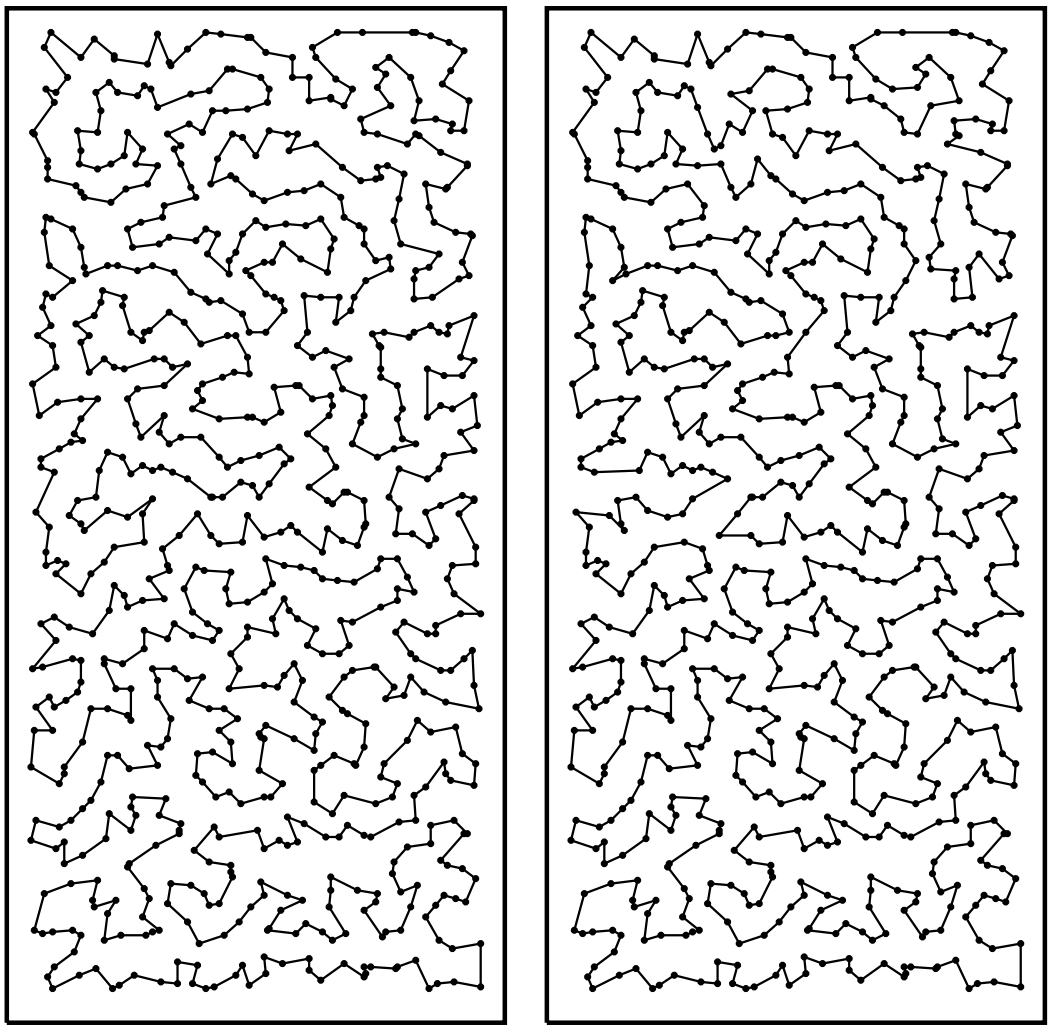
SPOT was implemented in two well-known NP-Complete problem domains, the TSP and the FSP. Experiments were carried out in the two domains for benchmarking purpose. The generated heuristics were non-dominated and competitive in the FSP domain in HyFlex, while the improvements in the TSP domain were limited. The results in both domains successfully validated the compatibility predicted. To verify the long-time result of using SPOT, one of the winners of the international hyper-heuristic competition CHeSC 2011, named PHunter, was tested with the generated heuristics. In TSP, the heuristics generated by the SPOT method gave unstable improvements in terms of solution quality. In FSP, the improvements were significant, and the score of PHunter was approximately doubled (from 20.5 to 43 with 50 maximum) and became the highest among all the leading hyper-heuristics. This is evidence on the success of SPOT.

There are two main contributions from this research. The first is the SPOT heuristic generation methodology, and the other is the enhancement

of definitions of the hyper-heuristics and its two subclasses. In addition, the capability of discovering instance-specific information from suboptima of subproblems is an interesting property. It was also the key to the success of the SPOT heuristic generation. SPOT can be applied in any combinatorial optimization domain if the problem instances in the domain meet the standard requirements and the compatibility measures. Therefore, SPOT is a conditionally cross-domain algorithm and its new property might probably be an answer to the issues raised by the NFL theorems.

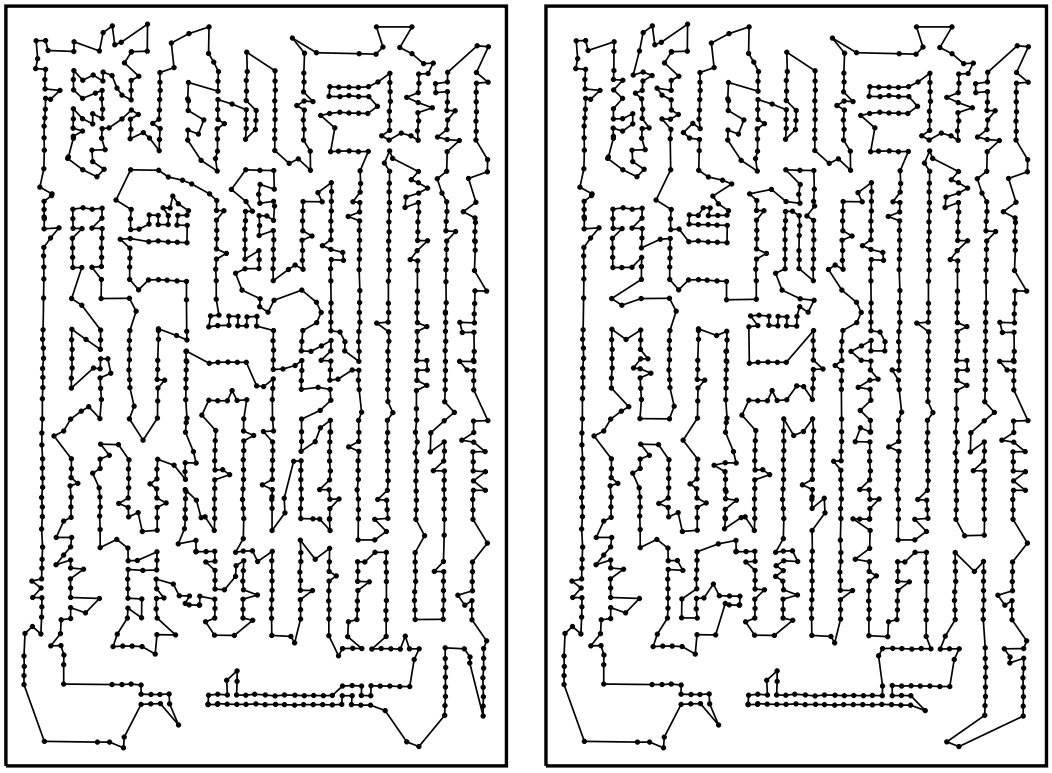
Future works on SPOT and related concepts include continued research on the non-stochastic machine learning techniques and the refinements of the definitions. It would also be meaningful to modify the-state-of-the-art LLHs by employing SPOT to push forward the Pareto frontier for general hyper-heuristics in terms of performance. Another interesting research could be making a transformation map. On the map, the distances and paths of transformations from well-known combinatorial optimization problems to the required models of SPOT can be drawn, so that the adaption of SPOT will be convenient in practice.

Appendix A Suboptima of Training Instances



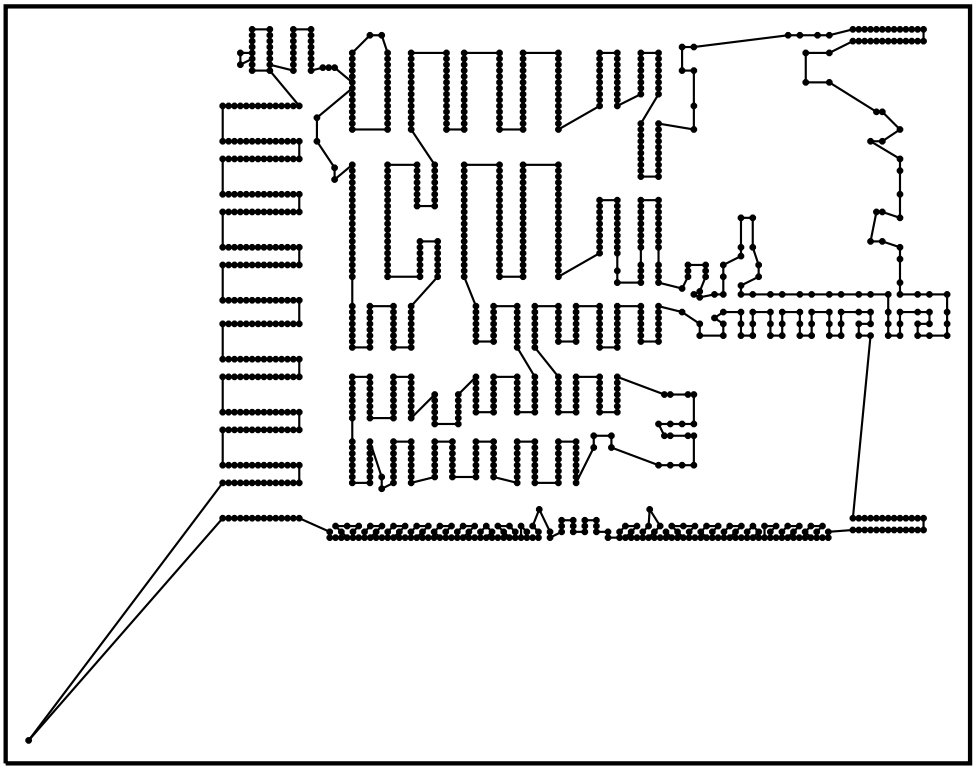
(i) Inst 4, Cost = 8873.002687969361 (ii) Inst 4, Cost = 8877.2061295038

Figure A.1: Suboptima of the training instance `rat783` in the TSP

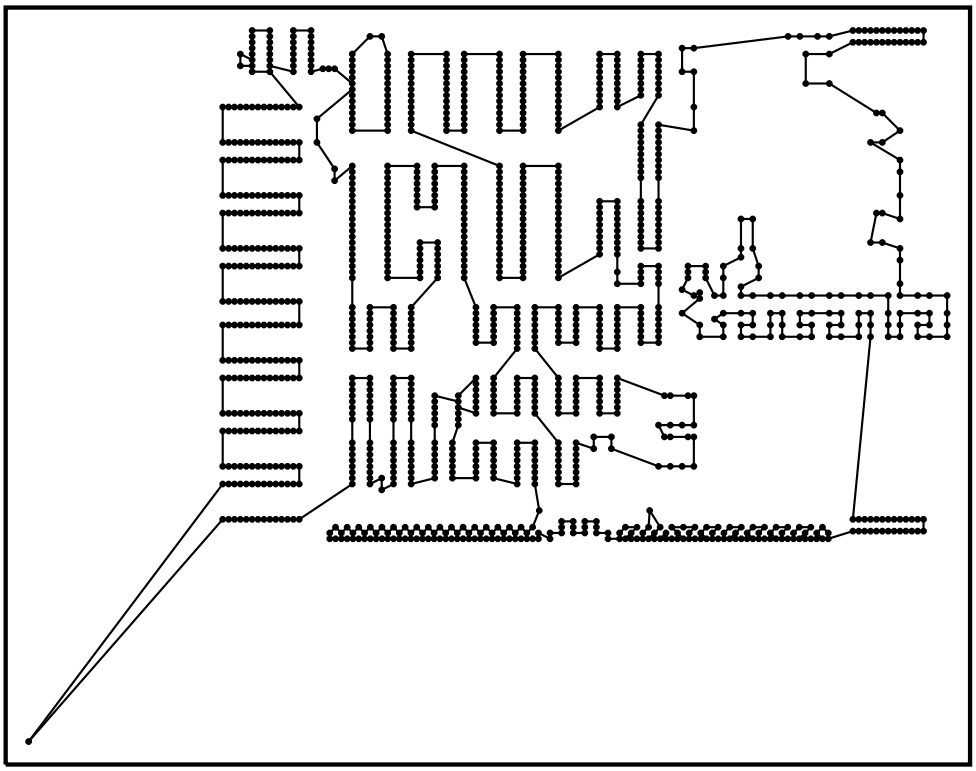


(i) Inst 5, Cost = 57210.04615291555 (ii) Inst 5, Cost = 57142.007438059256

Figure A.2: Suboptima of the training instance `pcb1173` in the TSP



(i) Inst 6, Cost = 52288.38373274874



(ii) Inst 6, Cost = 52446.80109412457

Figure A.3: Suboptima of the training instance `d1291` in the TSP

Table A.1: Suboptima of the training instances in the FSP

Instance (C_{max})	Solution
Inst 0 (6265)	53, 77, 58, 21, 81, 0, 2, 39, 78, 75, 50, 34, 73, 3, 82, 32, 80, 61, 4, 46, 20, 5, 27, 62, 88, 55, 30, 98, 59, 24, 85, 45, 15, 69, 11, 84, 57, 49, 1, 43, 65, 60, 52, 79, 10, 99, 8, 54, 90, 9, 93, 29, 38, 19, 71, 42, 92, 31, 66, 47, 41, 76, 36, 95, 14, 68, 33, 16, 44, 64, 94, 28, 25, 70, 63, 86, 56, 17, 23, 67, 89, 87, 51, 22, 37, 13, 72, 97, 18, 74, 7, 96, 91, 35, 83, 26, 48, 12, 6, 40
Inst 0 (6269)	53, 21, 46, 0, 34, 82, 54, 4, 79, 81, 73, 58, 50, 78, 30, 39, 55, 80, 20, 38, 88, 11, 99, 2, 49, 64, 75, 32, 52, 47, 60, 15, 69, 70, 8, 9, 10, 42, 43, 3, 59, 65, 44, 5, 84, 94, 92, 31, 27, 61, 29, 90, 14, 36, 17, 85, 77, 33, 1, 93, 35, 72, 98, 24, 41, 19, 86, 22, 16, 95, 28, 23, 37, 66, 76, 62, 45, 97, 71, 67, 87, 63, 74, 91, 7, 96, 89, 51, 18, 56, 13, 25, 57, 83, 26, 68, 48, 12, 6, 40
Inst 1 (6238)	49, 48, 4, 94, 67, 30, 75, 51, 0, 65, 96, 83, 9, 89, 34, 68, 64, 35, 63, 99, 74, 61, 81, 53, 26, 22, 2, 13, 12, 95, 18, 77, 98, 84, 8, 39, 59, 10, 71, 32, 19, 7, 46, 88, 15, 50, 21, 58, 41, 6, 82, 42, 47, 76, 3, 80, 78, 87, 31, 43, 1, 52, 40, 90, 62, 86, 24, 70, 25, 92, 29, 27, 36, 33, 85, 97, 69, 60, 72, 44, 57, 16, 5, 55, 73, 79, 37, 45, 14, 93, 11, 20, 66, 28, 91, 23, 17, 38, 56, 54
Inst 1 (6246)	48, 94, 64, 49, 65, 68, 18, 61, 30, 96, 77, 13, 32, 2, 99, 34, 95, 88, 67, 51, 53, 89, 42, 8, 81, 3, 1, 78, 9, 59, 52, 39, 75, 82, 19, 46, 90, 4, 6, 23, 15, 21, 27, 37, 31, 74, 76, 58, 83, 29, 73, 10, 22, 33, 97, 47, 92, 50, 45, 70, 0, 7, 35, 84, 14, 12, 79, 60, 25, 63, 26, 57, 69, 72, 44, 87, 43, 91, 85, 24, 55, 17, 40, 62, 86, 16, 41, 71, 36, 20, 66, 28, 5, 11, 54, 98, 93, 80, 38, 56
Inst 2 (6316)	86, 9, 91, 60, 65, 10, 63, 66, 75, 28, 50, 23, 41, 46, 85, 22, 47, 57, 53, 4, 55, 97, 11, 33, 21, 56, 19, 80, 68, 45, 32, 79, 69, 96, 12, 59, 74, 3, 52, 39, 84, 87, 40, 98, 77, 36, 24, 5, 93, 67, 14, 31, 15, 29, 76, 2, 27, 18, 48, 34, 95, 58, 16, 71, 20, 1, 26, 43, 30, 8, 42, 94, 89, 90, 7, 78, 62, 54, 88, 38, 92, 72, 83, 35, 25, 6, 70, 49, 0, 73, 51, 37, 61, 64, 44, 81, 17, 99, 13, 82
Inst 2 (6321)	86, 9, 40, 60, 10, 96, 63, 91, 66, 41, 57, 75, 85, 50, 28, 23, 46, 47, 62, 22, 53, 59, 55, 80, 45, 32, 97, 11, 21, 33, 79, 69, 4, 56, 38, 39, 92, 24, 3, 19, 68, 52, 65, 77, 84, 87, 5, 93, 29, 76, 14, 31, 67, 2, 27, 98, 18, 16, 48, 34, 95, 58, 71, 1, 20, 54, 26, 43, 30, 89, 42, 7, 49, 74, 8, 37, 78, 44, 36, 88, 72, 0, 83, 35, 73, 90, 25, 6, 70, 12, 64, 51, 61, 15, 81, 13, 82, 17, 94, 99
Inst 3 (6323)	0, 67, 44, 4, 43, 59, 78, 88, 73, 68, 21, 47, 45, 34, 87, 28, 37, 64, 2, 38, 85, 32, 75, 53, 95, 66, 54, 60, 15, 55, 70, 50, 84, 83, 25, 27, 91, 6, 90, 94, 76, 1, 72, 24, 8, 71, 57, 30, 18, 97, 14, 92, 62, 16, 58, 48, 99, 3, 10, 98, 7, 42, 89, 22, 69, 31, 23, 77, 17, 93, 49, 12, 86, 63, 11, 41, 82, 5, 80, 36, 56, 65, 61, 46, 51, 20, 33, 74, 40, 9, 81, 96, 29, 19, 13, 26, 52, 79, 35, 39
Inst 3 (6331)	0, 59, 28, 44, 4, 50, 68, 45, 43, 73, 15, 54, 87, 49, 56, 2, 67, 64, 10, 53, 95, 94, 32, 11, 88, 55, 71, 83, 1, 92, 78, 20, 24, 7, 37, 85, 57, 27, 72, 76, 98, 66, 91, 82, 69, 70, 77, 99, 97, 9, 62, 23, 38, 40, 34, 60, 12, 30, 86, 36, 17, 14, 31, 21, 22, 42, 18, 75, 25, 93, 89, 46, 84, 51, 3, 8, 90, 61, 29, 6, 16, 33, 5, 74, 19, 80, 65, 41, 52, 13, 58, 63, 96, 47, 48, 81, 26, 79, 35, 39

Appendix B Examples of Data and Results of Learning

Table B.1: The group and raw attributes of assignments in the 20-nearest-neighbor-first candidate set of two subproblems of the TSP instance `rat783` (15840 rows $\times$ (1 column of group + 2 columns of raw attributes), $n' = 400$, sampled by a random selection and a rectangular selection, respectively)

Group Id	A1 (distance)	A2 (angle)
316	13.04	0.08
	17.20	0.62
	23.09	1.26
	23.77	-1.32
	27.02	-0.89
	27.80	-0.66
	29.00	0.81
	30.41	0.48
	34.13	-1.02
	37.16	-0.42
	38.12	1.49
	40.20	-1.47
	40.79	0.20
	42.20	0.63
	43.46	0.40
	45.18	-0.09
	45.28	1.04
	45.49	0.99
	46.53	1.08
	48.51	-1.32
⋮	⋮	⋮
167	16.40	-0.66
	17.20	-2.19
	26.08	-2.14
	26.93	0.38
	28.32	2.41
	30.23	-0.60
	30.87	1.14
	36.24	2.06
	36.40	-0.37
	36.67	-2.95
	37.36	0.27
	40.45	2.99
	42.15	-2.75
	43.05	0.05
	43.86	1.15
	45.61	-2.48
	49.19	-0.46
	49.98	1.92
	50.64	-1.41
	51.09	2.27

Table B.2: The full training table consisting of normalized attributes of assignments and the label of suboptima of two subproblems of the TSP instance `rat783` (15840 rows $\times$ (86 columns of attributes + 1 column of labels), attributes normalized from Table B.1)

A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	$\cdots$	A85	A86	Label
13.04	20	0	0	1	0.76	0.56	0.55	0.48	0.47	$\cdots$	0.85	0.85	1
17.20	20	0	0	1.32	1	0.75	0.72	0.64	0.62	$\cdots$	0.85	0.85	0
23.09	20	0	0	1.77	1.34	1	0.97	0.85	0.83	$\cdots$	0.85	0.85	1
23.77	20	0	0	1.82	1.38	1.03	1	0.88	0.85	$\cdots$	0.85	0.85	0
27.02	20	0	0	2.07	1.57	1.17	1.14	1	0.97	$\cdots$	0.85	0.85	0
27.80	20	0	0	2.13	1.62	1.20	1.17	1.03	1	$\cdots$	0.85	0.85	0
29.00	20	0	0	2.22	1.69	1.26	1.22	1.07	1.04	$\cdots$	0.85	0.85	0
30.41	20	0	0	2.33	1.77	1.32	1.28	1.13	1.09	$\cdots$	0.85	0.85	0
34.13	20	0	0	2.62	1.98	1.48	1.44	1.26	1.23	$\cdots$	0.85	0.85	0
37.16	20	0	0	2.85	2.16	1.61	1.56	1.38	1.34	$\cdots$	0.85	0.85	0
38.12	20	0	0	2.92	2.22	1.65	1.60	1.41	1.37	$\cdots$	0.85	0.85	0
40.20	20	0	0	3.08	2.34	1.74	1.69	1.49	1.45	$\cdots$	0.85	0.85	0
40.79	20	0	0	3.13	2.37	1.77	1.72	1.51	1.47	$\cdots$	0.85	0.85	0
42.20	20	0	0	3.24	2.45	1.83	1.78	1.56	1.52	$\cdots$	0.85	0.85	0
43.46	20	0	0	3.33	2.53	1.88	1.83	1.61	1.56	$\cdots$	0.85	0.85	0
45.18	20	0	0	3.46	2.63	1.96	1.90	1.67	1.62	$\cdots$	0.85	0.85	0
45.28	20	0	0	3.47	2.63	1.96	1.90	1.68	1.63	$\cdots$	0.85	0.85	0
45.49	20	0	0	3.49	2.64	1.97	1.91	1.68	1.64	$\cdots$	0.85	0.85	0
46.53	20	0	0	3.57	2.70	2.02	1.96	1.72	1.67	$\cdots$	0.85	0.85	0
48.51	20	0	0	3.72	2.82	2.10	2.04	1.80	1.74	$\cdots$	0.85	0.85	0
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$
16.40	20	0	0	1	0.95	0.63	0.61	0.58	0.54	$\cdots$	0.94	0.80	1
17.20	20	0	0	1.05	1	0.66	0.64	0.61	0.57	$\cdots$	0.94	0.80	1
26.08	20	0	0	1.59	1.52	1	0.97	0.92	0.86	$\cdots$	0.94	0.80	0
26.93	20	0	0	1.64	1.57	1.03	1	0.95	0.89	$\cdots$	0.94	0.80	0
28.32	20	0	0	1.73	1.65	1.09	1.05	1	0.94	$\cdots$	0.94	0.80	0
30.23	20	0	0	1.84	1.76	1.16	1.12	1.07	1	$\cdots$	0.94	0.80	0
30.87	20	0	0	1.88	1.79	1.18	1.15	1.09	1.02	$\cdots$	0.94	0.80	0
36.24	20	0	0	2.21	2.11	1.39	1.35	1.28	1.20	$\cdots$	0.94	0.80	0
36.40	20	0	0	2.22	2.12	1.40	1.35	1.29	1.20	$\cdots$	0.94	0.80	0
36.67	20	0	0	2.24	2.13	1.41	1.36	1.30	1.21	$\cdots$	0.94	0.80	0
37.36	20	0	0	2.28	2.17	1.43	1.39	1.32	1.24	$\cdots$	0.94	0.80	0
40.45	20	0	0	2.47	2.35	1.55	1.50	1.43	1.34	$\cdots$	0.94	0.80	0
42.15	20	0	0	2.57	2.45	1.62	1.57	1.49	1.39	$\cdots$	0.94	0.80	0
43.05	20	0	0	2.62	2.50	1.65	1.60	1.52	1.42	$\cdots$	0.94	0.80	0
43.86	20	0	0	2.67	2.55	1.68	1.63	1.55	1.45	$\cdots$	0.94	0.80	0
45.61	20	0	0	2.78	2.65	1.75	1.69	1.61	1.51	$\cdots$	0.94	0.80	0
49.19	20	0	0	3.00	2.86	1.89	1.83	1.74	1.63	$\cdots$	0.94	0.80	0
49.98	20	0	0	3.05	2.91	1.92	1.86	1.76	1.65	$\cdots$	0.94	0.80	0
50.64	20	0	0	3.09	2.94	1.94	1.88	1.79	1.67	$\cdots$	0.94	0.80	0
51.09	20	0	0	3.11	2.97	1.96	1.90	1.80	1.69	$\cdots$	0.94	0.80	0

Table B.3: Results of the attribute selection for the full training table of Table B.2 (15840 rows × (8 columns of attributes + 1 column of labels))

A1	A3	A5	A6	A7	A8	A14	A20	Label
13.04	0	1	0.76	0.56	0.55	0.35	0.29	1
17.20	0	1.32	1	0.75	0.72	0.46	0.38	0
23.09	0	1.77	1.34	1	0.97	0.62	0.51	1
23.77	0	1.82	1.38	1.03	1	0.64	0.53	0
27.02	0	2.07	1.57	1.17	1.14	0.73	0.60	0
27.80	0	2.13	1.62	1.20	1.17	0.75	0.62	0
29.00	0	2.22	1.69	1.26	1.22	0.78	0.64	0
30.41	0	2.33	1.77	1.32	1.28	0.82	0.67	0
34.13	0	2.62	1.98	1.48	1.44	0.92	0.76	0
37.16	0	2.85	2.16	1.61	1.56	1	0.82	0
38.12	0	2.92	2.22	1.65	1.60	1.03	0.84	0
40.20	0	3.08	2.34	1.74	1.69	1.08	0.89	0
40.79	0	3.13	2.37	1.77	1.72	1.10	0.90	0
42.20	0	3.24	2.45	1.83	1.78	1.14	0.93	0
43.46	0	3.33	2.53	1.88	1.83	1.17	0.96	0
45.18	0	3.46	2.63	1.96	1.90	1.22	1	0
45.28	0	3.47	2.63	1.96	1.90	1.22	1.00	0
45.49	0	3.49	2.64	1.97	1.91	1.22	1.01	0
46.53	0	3.57	2.70	2.02	1.96	1.25	1.03	0
48.51	0	3.72	2.82	2.10	2.04	1.31	1.07	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
16.40	0	1	0.95	0.63	0.61	0.45	0.36	1
17.20	0	1.05	1	0.66	0.64	0.47	0.38	1
26.08	0	1.59	1.52	1	0.97	0.71	0.57	0
26.93	0	1.64	1.57	1.03	1	0.73	0.59	0
28.32	0	1.73	1.65	1.09	1.05	0.77	0.62	0
30.23	0	1.84	1.76	1.16	1.12	0.82	0.66	0
30.87	0	1.88	1.79	1.18	1.15	0.84	0.68	0
36.24	0	2.21	2.11	1.39	1.35	0.99	0.79	0
36.40	0	2.22	2.12	1.40	1.35	0.99	0.80	0
36.67	0	2.24	2.13	1.41	1.36	1	0.80	0
37.36	0	2.28	2.17	1.43	1.39	1.02	0.82	0
40.45	0	2.47	2.35	1.55	1.50	1.10	0.89	0
42.15	0	2.57	2.45	1.62	1.57	1.15	0.92	0
43.05	0	2.62	2.50	1.65	1.60	1.17	0.94	0
43.86	0	2.67	2.55	1.68	1.63	1.20	0.96	0
45.61	0	2.78	2.65	1.75	1.69	1.24	1	0
49.19	0	3.00	2.86	1.89	1.83	1.34	1.08	0
49.98	0	3.05	2.91	1.92	1.86	1.36	1.10	0
50.64	0	3.09	2.94	1.94	1.88	1.38	1.11	0
51.09	0	3.11	2.97	1.96	1.90	1.39	1.12	0

```
J48 pruned tree
------------------
A8 <= 0.986899
|   A6 <= 0.936329: 1 (711.0/69.0)
|   A6 > 0.936329
|   |   A7 <= 0.907959
|   |   |   A5 <= 1.425106
|   |   |   |   A8 <= 0.795966: 1 (308.0/86.0)
|   |   |   |   A8 > 0.795966: 0 (60.0/27.0)
|   |   |   A5 > 1.425106: 0 (229.0/106.0)
|   |   A7 > 0.907959: 0 (1007.0/299.0)
A8 > 0.986899: 0 (13525.0/288.0)

Number of Leaves  :         6
Size of the tree :         11
```

Figure B.1: Instance-specific result of the J48 classifier for the TSP instance `rat783`, training data shown in Table B.3

```
JRIP rules:
===========
1. (A7 <= 1.01303) and (A7 <= 0.823592) and (A6 <= 0.800735)
   => Label=1 (510.0/25.0)
2. (A7 <= 1.061977) and (A7 <= 0.906932) and (A5 <= 1.433721)
   and (A8 <= 0.720805) => Label=1 (389.0/90.0)
3. (A7 <= 1.126515) and (A8 <= 0.794645) and (A5 <= 1.355669)
   and (A6 <= 0.917267) => Label=1 (33.0/10.0)
4. (A8 <= 1.049303) and (A6 <= 1.059753) and (A8 <= 0.866758)
   and (A5 <= 1.566151) => Label=1 (319.0/154.0)
5. (A14 <= 0.692356) and (A6 <= 1.194349) and (A14 <= 0.504353)
   and (A7 <= 0.709541) => Label=1 (51.0/22.0)
6. (A8 <= 1.051403) and (A6 <= 1.155428) and (A14 <= 0.493171)
   and (A14 >= 0.479119) => Label=1 (38.0/16.0)
7. (A8 <= 1.051403) and (A7 <= 1.01716) and (A20 <= 0.385248)
   and (A5 <= 3.508232) and (A8 >= 0.938898) => Label=1
   (37.0/15.0)
8. => Label=0 (14463.0/539.0)

Number of Rules : 8
```

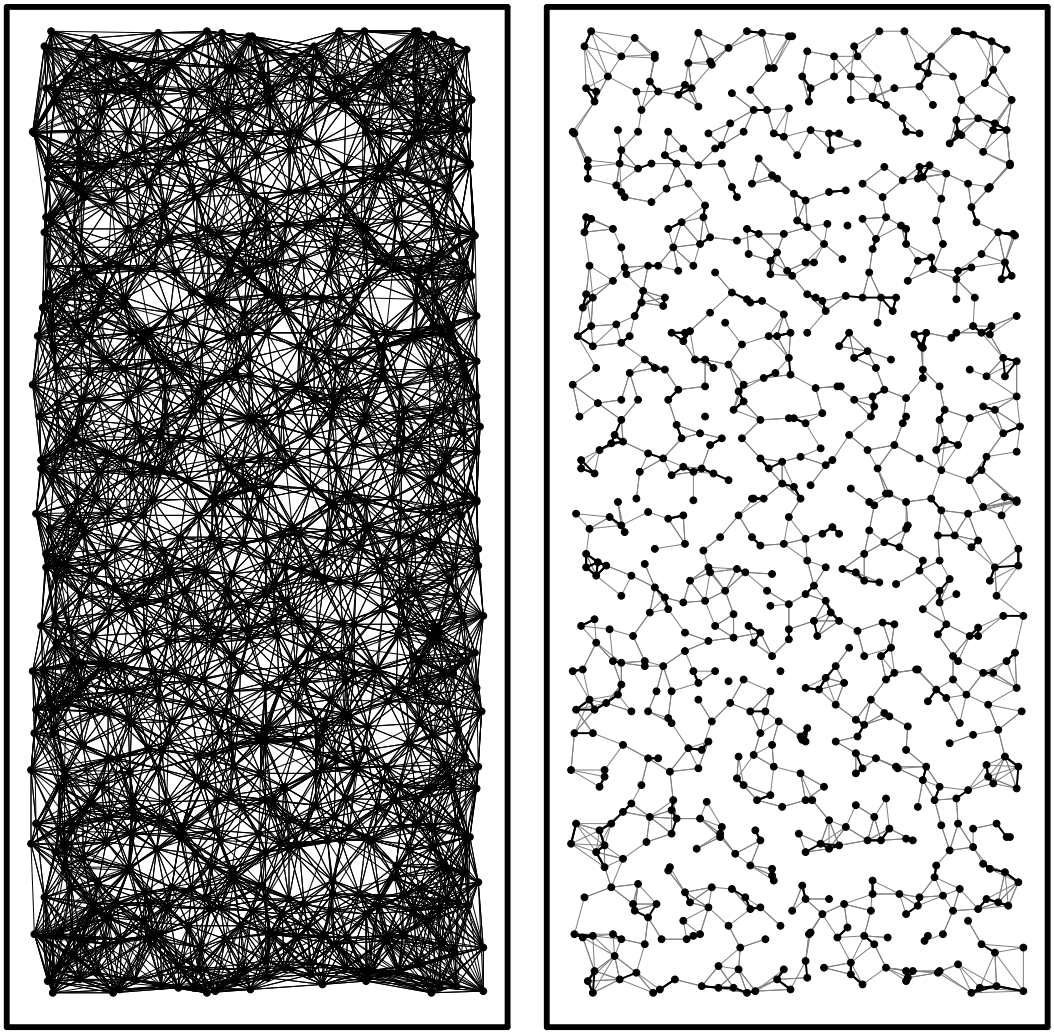
Figure B.2: Instance-specific result of the JRip classifier for the TSP instance `rat783`, training data shown in Table B.3

```
Naive Bayes Classifier

Class 0: Prior probability = 0.9
A1:  Normal Distribution. Mean = 34.1648 StandardDev = 12.2188
     WeightSum = 14256 Precision = 0.07893093700787401
A3:  Normal Distribution. Mean = 0 StandardDev = 0.1667
     WeightSum = 14256 Precision = 1.0
A5:  Normal Distribution. Mean = 4.6907 StandardDev = 5.3678
     WeightSum = 14256 Precision = 0.005596767589330649
A6:  Normal Distribution. Mean = 2.4956 StandardDev = 1.2026
     WeightSum = 14256 Precision = 0.0020412012114637015
A7:  Normal Distribution. Mean = 1.9844 StandardDev = 0.7277
     WeightSum = 14256 Precision = 7.003549220879403E-4
A8:  Normal Distribution. Mean = 1.6846 StandardDev = 0.5389
     WeightSum = 14256 Precision = 3.5252928054789944E-4
A14: Normal Distribution. Mean = 1.0454 StandardDev = 0.2923
     WeightSum = 14256 Precision = 1.501225443814196E-4
A20: Normal Distribution. Mean = 0.8162 StandardDev = 0.2232
     WeightSum = 14256 Precision = 1.0161365097366262E-4

Class 1: Prior probability = 0.1
A1:  Normal Distribution. Mean = 13.9825 StandardDev = 6.5628
     WeightSum = 1584 Precision = 0.07893093700787401
A3:  Normal Distribution. Mean = 0.0076 StandardDev = 0.1667
     WeightSum = 1584 Precision = 1.0
A5:  Normal Distribution. Mean = 1.675 StandardDev = 1.9162
     WeightSum = 1584 Precision = 0.005596767589330649
A6:  Normal Distribution. Mean = 0.9672 StandardDev = 0.4053
     WeightSum = 1584 Precision = 0.0020412012114637015
A7:  Normal Distribution. Mean = 0.7923 StandardDev = 0.3266
     WeightSum = 1584 Precision = 7.003549220879403E-4
A8:  Normal Distribution. Mean = 0.6861 StandardDev = 0.2872
     WeightSum = 1584 Precision = 3.5252928054789944E-4
A14: Normal Distribution. Mean = 0.4315 StandardDev = 0.1895
     WeightSum = 1584 Precision = 1.501225443814196E-4
A20: Normal Distribution. Mean = 0.3378 StandardDev = 0.149
     WeightSum = 1584 Precision = 1.0161365097366262E-4
```

Figure B.3: Instance-specific result of the NaiveBayes classifier for the TSP instance `rat783`, training data shown in Table B.3

Table B.4: Eight selected columns of test data and the summation of predicted labels of the assignments in the 20-nearest-neighbor-first candidate set of the TSP instance `rat783` (15660 rows × (8 columns of selected attributes + 1 column of summations of labels))

A1	A3	A5	A6	A7	A8	A14	A20	Σ label
7.62	0	1	0.58	0.43	0.30	0.21	0.14	2
13.04	0	1.71	1	0.73	0.51	0.36	0.24	2
17.89	0	2.35	1.37	1	0.70	0.49	0.33	2
25.55	0	3.36	1.96	1.43	1	0.71	0.46	1
28.64	0	3.76	2.20	1.60	1.12	0.79	0.52	0
33.54	0	4.40	2.57	1.88	1.31	0.93	0.61	0
34.00	0	4.46	2.61	1.90	1.33	0.94	0.62	0
36.00	0	4.73	2.76	2.01	1.41	0.99	0.65	0
36.24	0	4.76	2.78	2.03	1.42	1	0.66	0
36.69	0	4.82	2.81	2.05	1.44	1.01	0.67	0
37.00	0	4.86	2.84	2.07	1.45	1.02	0.67	0
40.05	0	5.26	3.07	2.24	1.57	1.11	0.73	0
42.45	0	5.57	3.26	2.37	1.66	1.17	0.77	0
49.65	0	6.52	3.81	2.78	1.94	1.37	0.90	0
52.01	0	6.83	3.99	2.91	2.04	1.44	0.95	0
55.01	0	7.22	4.22	3.08	2.15	1.52	1	0
55.80	0	7.33	4.28	3.12	2.18	1.54	1.01	0
57.22	0	7.51	4.39	3.20	2.24	1.58	1.04	0
58.18	0	7.64	4.46	3.25	2.28	1.61	1.06	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
2.00	0	1	0.22	0.10	0.09	0.06	0.04	2
9.22	0	4.61	1	0.44	0.42	0.29	0.18	2
20.88	0	10.44	2.26	1	0.95	0.65	0.41	1
21.93	0	10.97	2.38	1.05	1	0.69	0.43	1
27.17	0	13.58	2.95	1.30	1.24	0.85	0.53	0
30.81	0	15.40	3.34	1.48	1.40	0.97	0.60	0
31.02	0	15.51	3.36	1.49	1.41	0.97	0.61	0
31.14	0	15.57	3.38	1.49	1.42	0.98	0.61	0
31.89	0	15.95	3.46	1.53	1.45	1	0.63	0
32.00	0	16.00	3.47	1.53	1.46	1.00	0.63	0
34.89	0	17.44	3.78	1.67	1.59	1.09	0.68	0
40.22	0	20.11	4.36	1.93	1.83	1.26	0.79	0
41.05	0	20.52	4.45	1.97	1.87	1.29	0.81	0
46.49	0	23.24	5.04	2.23	2.12	1.46	0.91	0
47.00	0	23.50	5.10	2.25	2.14	1.47	0.92	0
50.99	0	25.50	5.53	2.44	2.32	1.60	1	0
52.01	0	26.00	5.64	2.49	2.37	1.63	1.02	0
53.01	0	26.50	5.75	2.54	2.42	1.66	1.04	0
53.37	0	26.68	5.79	2.56	2.43	1.67	1.05	0
55.57	0	27.78	6.03	2.66	2.53	1.74	1.09	0



(i) The 20-nearest-neighbor-first candidate set

(ii) Promising assignments classified by the SPOT heuristic generation

Figure B.4: A set of promising edges selected from the 20-nearest-neighbor-first candidate set of the training instance `rat783` in the TSP, according to the Σ label column in Table B.4 (where black edges in the second figure were the assignments with Σ label ≥ 2 and the gray edges were those with Σ label $= 1$)

Table B.5: The group and raw attributes of assignments of the "following" relationship in two subproblems of the FSP instance `TA082` (3480 rows $\times$ (1 column of group + 103 columns of raw attributes), $n' = 30$, sampled by two random selections, respectively)

Group	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	$\cdots$	A103
541	56.2	515.64	4555	-633	-0.02	96	41	64	82	59	30	64	45	44	$\cdots$	96
	47.2	759.82	4414	145	0.04	35	81	10	54	38	51	26	83	87	$\cdots$	89
567	56.2	515.64	5403	-633	-0.02	96	41	64	82	59	30	64	45	44	$\cdots$	96
	49.6	991.10	5502	49	-0.09	72	99	62	10	1	64	16	2	73	$\cdots$	99
562	56.2	515.64	5211	-633	-0.02	96	41	64	82	59	30	64	45	44	$\cdots$	96
	43.5	637.00	5415	637	0.06	40	40	36	22	57	87	8	23	5	$\cdots$	87
565	56.2	515.64	4075	-633	-0.02	96	41	64	82	59	30	64	45	44	$\cdots$	96
	44.5	602.47	4156	-1173	-0.08	25	56	49	75	59	21	76	36	53	$\cdots$	83
545	56.2	515.64	5635	-633	-0.02	96	41	64	82	59	30	64	45	44	$\cdots$	96
	42.2	694.03	5830	297	-0.03	31	19	50	3	75	31	40	8	77	$\cdots$	96
560	56.2	515.64	4808	-633	-0.02	96	41	64	82	59	30	64	45	44	$\cdots$	96
	50.8	754.91	5450	93	0.02	54	58	34	53	38	18	78	28	71	$\cdots$	95
566	56.2	515.64	5361	-633	-0.02	96	41	64	82	59	30	64	45	44	$\cdots$	96
	62.1	664.89	4824	342	0.02	59	29	32	87	67	93	21	85	84	$\cdots$	93
553	56.2	515.64	6629	-633	-0.02	96	41	64	82	59	30	64	45	44	$\cdots$	96
	51.8	933.36	6948	1401	-0.17	78	28	23	8	4	2	58	93	58	$\cdots$	97
544	56.2	515.64	5320	-633	-0.02	96	41	64	82	59	30	64	45	44	$\cdots$	96
	47.5	1043.95	6236	2466	0.09	5	6	25	62	7	80	57	25	10	$\cdots$	99
557	56.2	515.64	4437	-633	-0.02	96	41	64	82	59	30	64	45	44	$\cdots$	96
	52.1	727.67	5087	191	0.02	41	46	40	38	8	38	77	97	37	$\cdots$	97
551	56.2	515.64	4682	-633	-0.02	96	41	64	82	59	30	64	45	44	$\cdots$	96
	50.9	886.62	4498	-1528	-0.14	99	65	71	31	74	56	78	45	74	$\cdots$	99
547	56.2	515.64	2599	-633	-0.02	96	41	64	82	59	30	64	45	44	$\cdots$	96
	54.1	654.62	4189	-1031	-0.03	39	88	73	60	22	61	59	33	76	$\cdots$	95
559	56.2	515.64	4930	-633	-0.02	96	41	64	82	59	30	64	45	44	$\cdots$	96
	44.7	851.50	4997	-424	-0.02	89	71	13	97	14	35	40	26	54	$\cdots$	97
555	56.2	515.64	3974	-633	-0.02	96	41	64	82	59	30	64	45	44	$\cdots$	96
	48.5	955.21	4141	-752	-0.09	41	88	82	40	78	79	2	9	32	$\cdots$	97
554	56.2	515.64	5747	-633	-0.02	96	41	64	82	59	30	64	45	44	$\cdots$	96
	57.2	757.33	5611	176	-0.03	96	9	27	88	39	37	64	72	91	$\cdots$	96
540	56.2	515.64	4357	-633	-0.02	96	41	64	82	59	30	64	45	44	$\cdots$	96
	44.5	579.53	4315	507	-0.02	44	25	35	73	33	37	25	38	31	$\cdots$	96
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
331	45.4	888.34	4707	-369	-0.03	19	84	84	25	11	58	33	35	55	$\cdots$	97
	56.2	515.64	3098	-633	-0.02	96	41	64	82	59	30	64	45	44	$\cdots$	96

Table B.6: The full training table from two subproblems of the FSP instance `TA082` (3480 rows $\times$ (206 columns of attributes + 1 column of labels), attributes normalized from Table B.5)

A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12 $\cdots$	A206	Label
56.2	1.19	515.64	0.68	4555	1.03	-633	-4.37	-0.02	-0.44	96	2.74 $\cdots$	1.08	1
47.2	0.84	759.82	1.47	4414	0.97	145	-0.23	0.04	-2.28	35	0.36 $\cdots$	0.93	0
56.2	1.13	515.64	0.52	5403	0.98	-633	-12.92	-0.02	0.19	96	1.33 $\cdots$	0.97	1
49.6	0.88	991.10	1.92	5502	1.02	49	-0.08	-0.09	5.18	72	0.75 $\cdots$	1.03	0
56.2	1.29	515.64	0.81	5211	0.96	-633	-0.99	-0.02	-0.30	96	2.40 $\cdots$	1.10	1
43.5	0.77	637.00	1.24	5415	1.04	637	-1.01	0.06	-3.35	40	0.42 $\cdots$	0.91	0
56.2	1.26	515.64	0.86	4075	0.98	-633	0.54	-0.02	0.23	96	3.84 $\cdots$	1.16	0
44.5	0.79	602.47	1.17	4156	1.02	-1173	1.85	-0.08	4.38	25	0.26 $\cdots$	0.86	1
56.2	1.33	515.64	0.74	5635	0.97	-633	-2.13	-0.02	0.68	96	3.10 $\cdots$	1.00	1
42.2	0.75	694.03	1.35	5830	1.03	297	-0.47	-0.03	1.46	31	0.32 $\cdots$	1.00	0
56.2	1.11	515.64	0.68	4808	0.88	-633	-6.81	-0.02	-0.98	96	1.78 $\cdots$	1.01	1
50.8	0.90	754.91	1.46	5450	1.13	93	-0.15	0.02	-1.02	54	0.56 $\cdots$	0.99	0
56.2	0.91	515.64	0.78	5361	1.11	-633	-1.85	-0.02	-0.72	96	1.63 $\cdots$	1.03	1
62.1	1.10	664.89	1.29	4824	0.90	342	-0.54	0.02	-1.39	59	0.61 $\cdots$	0.97	0
56.2	1.09	515.64	0.55	6629	0.95	-633	-0.45	-0.02	0.11	96	1.23 $\cdots$	0.99	1
51.8	0.92	933.36	1.81	6948	1.05	1401	-2.21	-0.17	9.50	78	0.81 $\cdots$	1.01	0
56.2	1.18	515.64	0.49	5320	0.85	-633	-0.26	-0.02	-0.19	96	19.20 $\cdots$	0.97	1
47.5	0.85	1043.95	2.02	6236	1.17	2466	-3.90	0.09	-5.18	5	0.05 $\cdots$	1.03	0
56.2	1.08	515.64	0.71	4437	0.87	-633	-3.31	-0.02	-0.81	96	2.34 $\cdots$	0.99	1
52.1	0.93	727.67	1.41	5087	1.15	191	-0.30	0.02	-1.24	41	0.43 $\cdots$	1.01	0
56.2	1.10	515.64	0.58	4682	1.04	-633	0.41	-0.02	0.12	96	0.97 $\cdots$	0.97	1
50.9	0.91	886.62	1.72	4498	0.96	-1528	2.41	-0.14	8.14	99	1.03 $\cdots$	1.03	0
56.2	1.04	515.64	0.79	2599	0.62	-633	0.61	-0.02	0.56	96	2.46 $\cdots$	1.01	0
54.1	0.96	654.62	1.27	4189	1.61	-1031	1.63	-0.03	1.78	39	0.41 $\cdots$	0.99	1
56.2	1.26	515.64	0.61	4930	0.99	-633	1.49	-0.02	0.86	96	1.08 $\cdots$	0.99	1
44.7	0.79	851.50	1.65	4997	1.01	-424	0.67	-0.02	1.16	89	0.93 $\cdots$	1.01	0
56.2	1.16	515.64	0.54	3974	0.96	-633	0.84	-0.02	0.19	96	2.34 $\cdots$	0.99	1
48.5	0.86	955.21	1.85	4141	1.04	-752	1.19	-0.09	5.18	41	0.43 $\cdots$	1.01	0
56.2	0.98	515.64	0.68	5747	1.02	-633	-3.60	-0.02	0.63	96	1.00 $\cdots$	1.00	1
57.2	1.02	757.33	1.47	5611	0.98	176	-0.28	-0.03	1.58	96	1.00 $\cdots$	1.00	0
56.2	1.26	515.64	0.89	4357	1.01	-633	-1.25	-0.02	0.84	96	2.18 $\cdots$	1.00	1
44.5	0.79	579.53	1.12	4315	0.99	507	-0.80	-0.02	1.19	44	0.46 $\cdots$	1.00	0
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$ $\ddots$	$\vdots$	$\vdots$
45.4	0.81	888.34	1.72	4707	1.52	-369	0.58	-0.03	1.58	19	0.20 $\cdots$	1.01	1
56.2	1.24	515.64	0.58	3098	0.66	-633	1.72	-0.02	0.63	96	5.05 $\cdots$	0.99	0

```
J48 pruned tree
------------------
A89 <= 32
|   A90 <= 0.685864: 1 (274.0)
|   A90 > 0.685864
|   |   A92 <= 0.876712: 1 (8.0)
|   |   A92 > 0.876712: 0 (8.0)
A89 > 32
|   A159 <= 8239
|   |   A160 <= 0.975358: 0 (326.0)
|   |   A160 > 0.975358
|   |   |   A30 <= 1.054054: 1 (14.0)
|   |   |   A30 > 1.054054: 0 (8.0)
|   A159 > 8239
|   |   A90 <= 3.648649
|   |   |   A166 <= 4.821429
|   |   |   |   A14 <= 7.333334
|   |   |   |   |   A189 <= 6255
|   |   |   |   |   |   A40 <= 19.4
|   |   |   |   |   |   |   A16 <= 6.857143
|   |   |   |   |   |   |   |   A13 <= 9
|   |   |   |   |   |   |   |   |   A94 <= 1.321951
|   |   |   |   |   |   |   |   |   |   A10 <= 1.282051: 1 (78.0)
|   |   |   |   |   |   |   |   |   |   A10 > 1.282051
|   |   |   |   |   |   |   |   |   |   |   A2 <=1.148707: 0(2.0)
|   |   |   |   |   |   |   |   |   |   |   A2 > 1.148707: 1(4.0)
|   |   |   |   |   |   |   |   |   A94 > 1.321951
|   |   |   |   |   |   |   |   |   |   A2 <= 1.174407: 0 (4.0)
|   |   |   |   |   |   |   |   |   |   A2 > 1.174407: 1 (4.0)
|   |   |   |   |   |   |   |   A13 > 9
|   |   |   |   |   |   |   |   |   A13 <= 20
|   |   |   |   |   |   |   |   |   |   A160 <= 1.206111
...
|   |   A90 > 3.648649
|   |   |   A34 <= 1.36: 0 (166.0)
|   |   |   A34 > 1.36
|   |   |   |   A16 <= 18: 0 (4.0)
|   |   |   |   A16 > 18: 1 (6.0)

Number of Leaves  :         157
Size of the tree :            313
```

Figure B.5: Instance-specific result of the J48 classifier for the FSP instance TA082, training data shown in Table B.6

```
JRIP rules:
===========
1.  (A92 <= 0.882979) and (A169 >= 439) and (A21 <= 64) and
    (A68 >= 0.910891) => Label=1 (404.0/0.0)
2.  (A134 >= 1.336439) and (A92 <= 1.11976) and (A185 >= 3581)
    and (A20 >= 0.946429) and (A68 <= 1.568389) and (A182 >=
    0.954958) => Label=1 (108.0/2.0)
3.  (A134 >= 1.061898) and (A134 >= 1.47425) and (A46 <=
    2.333333) and (A182 >= 1.016753) and (A40 >= 0.9125) =>
    Label=1 (124.0/10.0)
4.  (A94 <= 1.030769) and (A59 >= 234) and (A100 <= 0.862408)
    and (A90 <= 0.874074) and (A3 <= 754.905263) => Label=1
    (134.0/2.0)
5.  (A98 <= 1.091153) and (A134 >= 0.941898) and (A168 >=
    1.716763) and (A26 >= 0.428571) and (A206 <= 1.021277) =>
    Label=1 (106.0/8.0)
6.  (A134 >= 0.950803) and (A94 <= 1.27027) and (A181 >= 2618)
    and (A178 <= 1.158451) and (A132 >= 1.167996) and (A53 <=
    174) => Label=1 (68.0/8.0)
7.  (A92 <= 1.140625) and (A135 >= 3308) and (A69 <= 469) and (A80
    >= 0.840686) and (A168 <= 1.143382) => Label=1 (100.0/2.0)
8.  (A131 >= 2506) and (A92 <= 1.310606) and (A24 >= 1.24359)
    and (A129 >= 2091) and (A14 <= 1.107143) and (A26 >= 0.857143)
    => Label=1 (72.0/0.0)
9.  (A54 >= 0.843931) and (A96 <= 1.398104) and (A173 >= 942) and
    (A166 <= 0.887446) and (A137 <= 5502) and (A42 >= 0.545455)
    => Label=1 (52.0/0.0)
10. (A136 >= 1.147994) and (A98 <= 1.246201) and (A46 <= 1.769231)
    and (A7 >= -53) and (A108 <= 1.046472) and (A70 >= 0.778243)
    => Label=1 (66.0/2.0)
11. (A55 >= 192) and (A102 <= 1.219638) and (A23 >= 33) and (A4
    <= 1.288403) and (A186 >= 0.99223) and (A186 <= 1.252018) and
    (A3 >= 754.905263) and (A14 <= 4.4) => Label=1 (64.0/4.0)
12. (A96 <= 1.180995) and (A37 <= 56) and (A96 <= 0.805825) and
    (A10 >= 0.21875) and (A50 <= 1.264706) and (A60 >= 0.621212)
    => Label=1 (52.0/2.0)
13. (A134 >= 1.069912) and (A10 <= -0.368421) and (A16 >= 1.8)
    and (A182 <= 0.923543) => Label=1 (26.0/0.0)
14. (A181 >= 2642) and (A181 <= 2957) and (A92 <= 1.126984) and
    (A3 <= 914.681579) and (A31 >= 79) and (A194 >= 0.960687) =>
    Label=1 (54.0/0.0)
...
41. => Label=0 (1690.0/0.0)

Number of Rules : 41
```

Figure B.6: Instance-specific result of the JRip classifier for the FSP instance TA082, training data shown in Table B.6

```
Naive Bayes Classifier

Class 0: Prior probability = 0.5
A1:  Normal Distribution. Mean = 50.4556 StandardDev = 5.2841
     WeightSum = 1740 Precision = 0.4265306122448979
A2:  Normal Distribution. Mean = 1.0188 StandardDev = 0.1557
     WeightSum = 1740 Precision = 4.8687212103746397E-4
A3:  Normal Distribution. Mean = 845.4133 StandardDev = 165.8667
     WeightSum = 1740 Precision = 14.698139655172413
A4:  Normal Distribution. Mean = 1.1252 StandardDev = 0.341
     WeightSum = 1740 Precision = 0.0013090101322599194
A5:  Normal Distribution. Mean = 5581.2359 StandardDev = 1062.31
     WeightSum = 1740 Precision = 5.098496240601504
A6:  Normal Distribution. Mean = 1.0177 StandardDev = 0.2476
     WeightSum = 1740 Precision = 9.092975431530495E-4
A7:  Normal Distribution. Mean = 545.3688 StandardDev = 803.2476
     WeightSum = 1740 Precision = 81.51020408163265
A8:  Normal Distribution. Mean = 0.2234 StandardDev = 5.6679
     WeightSum = 1740 Precision = 0.05582411059757479
...
A206: Normal Distribution. Mean = 1.0171 StandardDev = 0.0649
     WeightSum = 1740 Precision = 0.0015540015384615388

Class 1: Prior probability = 0.5
A1:  Normal Distribution. Mean = 50.1061 StandardDev = 5.4521
     WeightSum = 1740 Precision = 0.4265306122448979
A2:  Normal Distribution. Mean = 1.0045 StandardDev = 0.1532
     WeightSum = 1740 Precision = 4.8687212103746397E-4
A3:  Normal Distribution. Mean = 792.9393 StandardDev = 190.5492
     WeightSum = 1740 Precision = 14.698139655172413
A4:  Normal Distribution. Mean = 0.9749 StandardDev = 0.3073
     WeightSum = 1740 Precision = 0.0013090101322599194
A5:  Normal Distribution. Mean = 5633.6625 StandardDev =1024.5525
     WeightSum = 1740 Precision = 5.098496240601504
A6:  Normal Distribution. Mean = 1.0401 StandardDev = 0.2506
     WeightSum = 1740 Precision = 9.092975431530495E-4
A7:  Normal Distribution. Mean = -31.8546 StandardDev = 677.081
     WeightSum = 1740 Precision = 81.51020408163265
A8:  Normal Distribution. Mean = 0.0055 StandardDev = 4.1143
     WeightSum = 1740 Precision = 0.05582411059757479
...
A206: Normal Distribution. Mean = 0.9873 StandardDev = 0.061
     WeightSum = 1740 Precision = 0.0015540015384615388
```

Figure B.7: Instance-specific result of the NaiveBayes classifier for the FSP instance `TA082`, training data shown in Table B.6

Table B.7: The test data and the summation of predicted labels of the assignments of the "following" relationship in the FSP instance `TA082` (19800 rows $\times$ (206 columns of normalized attributes + 1 column of summations of labels))

A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	$\cdots$	A206	Σ label
56.9	1.07	1065.25	1.08	5215	0.74	766	1.42	0.03	1.74	20	0.83	$\cdots$	1.02	2
53.3	0.94	984.54	0.92	7033	1.35	541	0.71	0.02	0.57	24	1.20	$\cdots$	0.98	2
56.9	1.04	1065.25	1.79	6440	1.04	766	1.55	0.03	1.19	20	0.50	$\cdots$	1.11	0
54.6	0.96	596.05	0.56	6187	0.96	494	0.64	0.03	0.84	40	2.00	$\cdots$	0.90	3
56.9	1.21	1065.25	1.23	5728	0.81	766	-2.61	0.03	-1.29	20	0.56	$\cdots$	1.03	0
47.0	0.83	865.00	0.81	7071	1.23	-293	-0.38	-0.03	-0.78	36	1.80	$\cdots$	0.97	3
56.9	1.35	1065.25	1.53	6575	1.28	766	2.58	0.03	-1.26	20	0.65	$\cdots$	1.03	3
42.2	0.74	694.03	0.65	5154	0.78	297	0.39	-0.03	-0.79	31	1.55	$\cdots$	0.97	0
56.9	1.38	1065.25	1.67	5751	0.96	766	-1.24	0.03	-0.42	20	0.43	$\cdots$	1.22	0
41.2	0.72	638.56	0.60	6012	1.05	-616	-0.80	-0.08	-2.38	46	2.30	$\cdots$	0.82	3
56.9	0.92	1065.25	1.60	3089	0.57	766	2.24	0.03	1.32	20	0.34	$\cdots$	1.06	0
62.1	1.09	664.89	0.62	5442	1.76	342	0.45	0.02	0.76	59	2.95	$\cdots$	0.94	3
56.9	0.93	1065.25	1.89	6380	1.08	766	1.99	0.03	1.90	20	0.71	$\cdots$	1.08	0
61.3	1.08	562.85	0.53	5896	0.92	384	0.50	0.02	0.53	28	1.40	$\cdots$	0.93	2
56.9	1.16	1065.25	1.28	7915	1.35	766	-6.03	0.03	-0.42	20	0.20	$\cdots$	1.00	1
49.1	0.86	832.68	0.78	5850	0.74	-127	-0.17	-0.08	-2.38	99	4.95	$\cdots$	1.00	2
56.9	1.04	1065.25	1.69	7147	1.42	766	0.48	0.03	1.68	20	2.86	$\cdots$	1.06	3
54.9	0.96	630.62	0.59	5037	0.70	1609	2.10	0.02	0.60	7	0.35	$\cdots$	0.94	1
56.9	0.96	1065.25	1.75	3966	0.78	766	-1.52	0.03	-1.45	20	0.27	$\cdots$	1.01	2
59.2	1.04	609.12	0.57	5096	1.28	-503	-0.66	-0.02	-0.69	74	3.70	$\cdots$	0.99	2
56.9	1.31	1065.25	1.29	6154	0.76	766	-0.43	0.03	-0.19	20	0.24	$\cdots$	1.03	0
43.6	0.77	828.46	0.78	8064	1.31	-1787	-2.33	-0.17	-5.17	84	4.20	$\cdots$	0.97	3
56.9	1.16	1065.25	1.78	6223	1.16	766	2.61	0.03	1.19	20	1.54	$\cdots$	1.06	1
49.3	0.87	596.93	0.56	5386	0.87	294	0.38	0.03	0.84	13	0.65	$\cdots$	0.94	2
56.9	1.31	1065.25	1.67	5461	1.12	766	1.20	0.03	0.55	20	0.50	$\cdots$	1.14	1
43.5	0.76	637.00	0.60	4873	0.89	637	0.83	0.06	1.82	40	2.00	$\cdots$	0.88	2
56.9	1.23	1065.25	1.06	7010	1.37	766	9.01	0.03	-1.10	20	0.29	$\cdots$	1.04	2
46.4	0.82	1006.88	0.95	5100	0.73	85	0.11	-0.03	-0.91	68	3.40	$\cdots$	0.96	1
56.9	1.17	1065.25	1.45	6639	0.88	766	-0.84	0.03	-1.45	20	0.24	$\cdots$	1.00	0
48.5	0.85	733.52	0.69	7518	1.13	-917	-1.20	-0.02	-0.69	85	4.25	$\cdots$	1.00	3
56.9	1.06	1065.25	1.19	6758	0.89	766	-0.89	0.03	-1.29	20	0.57	$\cdots$	1.01	0
53.8	0.95	896.48	0.84	7622	1.13	-861	-1.12	-0.03	-0.78	35	1.75	$\cdots$	0.99	3
56.9	1.05	1065.25	1.63	6415	1.07	766	-0.74	0.03	-1.03	20	0.51	$\cdots$	1.04	0
54.1	0.95	654.62	0.61	6015	0.94	-1031	-1.35	-0.03	-0.97	39	1.95	$\cdots$	0.96	3
56.9	1.01	1065.25	1.41	7126	1.40	766	0.55	0.03	1.00	20	1.82	$\cdots$	1.08	2
56.1	0.99	755.67	0.71	5082	0.71	1396	1.82	0.03	1.00	11	0.55	$\cdots$	0.93	1
56.9	1.08	1065.25	0.79	6098	1.17	766	6.23	0.03	-1.23	20	0.25	$\cdots$	1.01	1
52.7	0.93	1345.08	1.26	5217	0.86	123	0.16	-0.03	-0.82	79	3.95	$\cdots$	0.99	3
56.9	1.01	1065.25	1.23	5634	0.79	766	-0.34	0.03	-0.68	20	0.34	$\cdots$	1.00	1
56.3	0.99	867.25	0.81	7151	1.27	-2264	-2.96	-0.05	-1.48	58	2.90	$\cdots$	1.00	3
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
50.6	1.20	784.36	0.97	6265	1.06	1551	-21.85	0.03	0.58	51	4.64	$\cdots$	1.03	0
42.3	0.84	812.22	1.04	5933	0.95	-71	-0.05	0.05	1.71	11	0.22	$\cdots$	0.97	3

Table B.8: The weights (or summations of Σ label) according to Table B.7 (which generated the figure (i) in Figure 6.2)

Job Id	Weight by the SPOT	Job Id	Weight by the SPOT	Job Id	Weight by the SPOT
0	104	35	84	70	120
1	85	36	244	71	83
2	143	37	227	72	210
3	187	38	291	73	247
4	48	39	197	74	18
5	252	40	255	75	40
6	199	41	206	76	251
7	155	42	27	77	196
8	118	43	207	78	71
9	28	44	236	79	197
10	138	45	174	80	259
11	273	46	42	81	110
12	119	47	110	82	202
13	76	48	6	83	134
14	100	49	3	84	221
15	257	50	215	85	244
16	251	51	100	86	171
17	270	52	122	87	265
18	70	53	198	88	66
19	184	54	292	89	48
20	281	55	262	90	190
21	192	56	294	91	241
22	179	57	222	92	160
23	264	58	54	93	277
24	232	59	181	94	123
25	102	60	192	95	192
26	87	61	48	96	136
27	224	62	225	97	77
28	249	63	148	98	106
29	129	64	81	99	39
30	58	65	165		
31	109	66	166		
32	100	67	40		
33	103	68	11		
34	81	69	85		

Table B.9: An example of adding the weights of a predicted permutation and an NEH permutation (where the three columns of weights present the figures (i), (ii) and (iii) in Figure 6.2, respectively)

Job	Weight by			Job	Weight by			Job	Weight by		
Id	SPOT	NEH	Sum	Id	SPOT	NEH	Sum	Id	SPOT	NEH	Sum
0	32	69	101	35	23	71	94	70	39	34	73
1	24	72	96	36	80	62	142	71	22	26	48
2	46	64	110	37	76	43	119	72	70	29	99
3	57	81	138	38	97	97	194	73	82	83	165
4	10	3	13	39	63	70	133	74	3	23	26
5	86	87	173	40	87	91	178	75	8	2	10
6	66	49	115	41	68	52	120	76	85	42	127
7	48	65	113	42	4	14	18	77	62	50	112
8	37	67	104	43	69	82	151	78	17	30	47
9	5	25	30	44	78	89	167	79	64	63	127
10	45	92	137	45	53	21	74	80	89	10	99
11	94	76	170	46	9	27	36	81	36	93	129
12	38	55	93	47	35	37	72	82	67	58	125
13	18	35	53	48	1	0	1	83	43	53	96
14	27	47	74	49	0	1	1	84	72	60	132
15	88	38	126	50	71	75	146	85	81	90	171
16	84	85	169	51	29	59	88	86	52	79	131
17	93	73	166	52	40	88	128	87	92	54	146
18	16	36	52	53	65	84	149	88	15	77	92
19	56	95	151	54	98	17	115	89	12	13	25
20	96	96	192	55	90	80	170	90	58	28	86
21	59	57	116	56	99	99	198	91	79	86	165
22	54	48	102	57	73	15	88	92	49	33	82
23	91	19	110	58	13	22	35	93	95	74	169
24	77	66	143	59	55	45	100	94	41	18	59
25	30	44	74	60	60	41	101	95	61	6	67
26	26	11	37	61	11	56	67	96	44	39	83
27	74	61	135	62	75	31	106	97	19	8	27
28	83	98	181	63	47	78	125	98	33	94	127
29	42	40	82	64	21	4	25	99	6	7	13
30	14	5	19	65	50	16	66				
31	34	20	54	66	51	68	119				
32	28	24	52	67	7	9	16				
33	31	46	77	68	2	12	14				
34	20	32	52	69	25	51	76				

Appendix C Original Data of Figures and Tables

Table C.1: Original data of Figure 1.4 (D. S. Johnson & McGeoch, 2002, p. 376)

Heuristic	% excess over the HK bound	Normalized time (s)
Spacefill	34.56	0.02
Strip	30.75	0.01
Karp-20	29.34	0.85
NI	26.50	1.71
NN	24.79	0.28
CHCI	20.73	0.83
Greedy	16.42	0.20
FI	13.35	2.59
Savings	12.03	0.24
CCA	11.73	1129
AppChristo	11.05	0.44
Christo-S	9.81	1.04
GENI-10	5.89	823
2-Opt-JM	4.70	1.41
3-Opt-JM	2.88	1.50
LK-JM	2.00	2.06
Tabu-SC-SC	1.48	18830
MLLK-.1N	1.18	12.75
CLK-ABCC-N	0.90	63.91
Helsgaun.1N	0.69	1840

Table C.2: Original data of Figure 5.1

n'	cov (%)	$\bar{d}$	$\text{Time}_{\text{overall}}$ (s)	Time_{L} (s)
25	98.462	2.006	6.10	0.09
50	98.725	1.963	6.15	0.14
100	98.646	1.908	6.23	0.22
200	98.612	1.876	6.42	0.40
400	98.616	1.867	6.92	0.85
800	98.619	1.859	8.51	2.34
8-NN†	98.477	1.842	—	—

†: The 8-nearest-neighbor-first candidate set in HyFlex, with no modification by SPOT algorithms.

Table C.3: Original data of Table 5.10

	PHunter$_{64bit}$					SPOT-PHunter				
	No. 0	No. 2	No. 6	No. 7	No. 8	No. 0	No. 2	No. 6	No. 7	No. 8
31	48194.9	6796.5	52510.6	66916.3	21221065.6	48194.9	6804.4	52950.5	67107.7	21429281.3
runs	48194.9	6813.4	52771.2	66689.6	21221065.6	48194.9	6801.5	52809.9	66790.8	21429281.3
	48194.9	6825.0	52590.9	66686.0	21221065.6	48194.9	6802.1	52950.5	66606.4	21419868.0
	48194.9	6799.4	52839.1	66841.1	21221065.6	48194.9	6807.9	52950.5	66790.8	21429281.3
	48194.9	6820.1	52904.3	66693.0	21221065.6	48194.9	6805.7	52827.3	66606.4	21419868.0
	48194.9	6808.4	52923.5	67025.8	21221065.6	48194.9	6802.3	53047.6	66606.4	21429281.3
	48194.9	6812.4	52538.9	66722.6	21221065.6	48194.9	6802.1	52950.5	66606.4	21419868.0
	48194.9	6824.0	53200.3	67073.3	21221065.6	48194.9	6803.4	52552.2	66606.4	21419868.0
	48194.9	6805.0	52777.0	66887.8	21221065.6	48194.9	6803.7	52613.4	66606.4	21419868.0
	48194.9	6813.3	52844.5	66704.0	21221065.6	48194.9	6804.6	52732.5	66728.0	21419868.0
	48194.9	6820.2	53000.0	67272.9	21221065.6	48194.9	6804.8	52950.5	66894.9	21402829.4
	48194.9	6817.2	53004.3	66700.4	21221065.6	48194.9	6803.4	52552.2	66790.8	21419868.0
	48194.9	6819.6	52760.6	66744.5	21221065.6	48194.9	6801.5	52950.5	66606.4	21429281.3
	48194.9	6797.5	52717.4	66749.6	21221065.6	48194.9	6801.5	53140.4	66790.8	20957289.4
	48194.9	6811.6	52544.4	66487.7	21221065.6	48194.9	6801.5	52706.7	66790.8	21419868.0
	48194.9	6810.3	52593.0	66699.4	21221065.6	48194.9	6804.6	52872.4	66606.4	21377588.8
	48194.9	6812.6	52830.1	66490.6	21221065.6	48194.9	6802.8	52950.5	66606.4	21429281.3
	48194.9	6809.9	52973.0	66655.8	21221065.6	48194.9	6801.5	52872.7	66790.8	21429281.3
	48194.9	6815.3	52877.4	66960.9	21221065.6	48194.9	6804.7	52641.9	66790.8	21419868.0
	48194.9	6805.0	52835.2	66910.3	21221065.6	48194.9	6802.1	52950.5	66393.5	21429281.3
	48194.9	6812.4	52636.8	66801.1	21221065.6	48194.9	6802.1	52970.9	66606.4	21419868.0
	48194.9	6801.6	52952.6	66930.1	21221065.6	48194.9	6803.7	53146.2	66606.4	21402829.4
	48194.9	6810.1	52874.6	66635.1	21221065.6	48194.9	6802.4	52732.5	66614.9	21377588.8
	48194.9	6829.8	52716.8	66978.5	21221065.6	48194.9	6802.4	52976.0	66614.9	21419868.0
	48194.9	6824.8	53130.0	67164.0	21221065.6	48194.9	6802.4	52872.7	66606.4	21419868.0
	48194.9	6814.8	52672.1	67072.4	21221065.6	48194.9	6803.7	52552.2	66606.4	21419868.0
	48194.9	6799.2	52693.7	67305.3	21221065.6	48194.9	6802.4	53152.4	66606.4	21429281.3
	48194.9	6818.3	52613.4	67100.0	21221065.6	48194.9	6802.1	52950.5	66790.8	21419868.0
	48194.9	6807.2	52919.8	66987.9	21221065.6	48194.9	6807.9	52796.1	66606.4	21429281.3
	48194.9	6823.2	52873.4	66760.7	21221065.6	48194.9	6804.9	52834.1	66606.4	21377588.8
	48194.9	6820.4	52976.5	66826.0	21221065.6	48194.9	6802.1	52634.3	66790.8	21402829.4
Min	48194.9	6796.5	52510.6	66487.7	21221065.6	48194.9	6801.5	52552.2	66393.5	20957289.4
Med	48194.9	6812.6	52835.2	66826.0	21221065.6	48194.9	6802.4	52872.7	66606.4	21419868.0
Max	48194.9	6829.8	53200.3	67305.3	21221065.6	48194.9	6807.9	53152.4	67107.7	21429281.3

One-way ANOVA†	No. 0	No. 2	No. 6	No. 7	No. 8
F	—‡	35.95	1.21	15.14	210.25
p	—	0.00	0.28	0.00	0.00

†: Factor: PHunter$_{64bit}$ = 1, SPOT-PHunter = 2.
‡: Exactly the same.

Table C.4: Original data of Figure 5.3

Source	Hyper-heuristic	Score on each test instance					Overall
	(alphabetical)	No. 0	No. 2	No. 6	No. 7	No. 8	score
Results† of CHeSC 2011 (on a 32-bit Java)	ACO-HH	0	0	2	2	0	4
	AdaptHH	6	8	3	5	10	32
	Ant-Q	0	0	0	0	0	0
	AVEG-Nep	0	0	0	0	0	0
	DynILS	1	1	0	0	4	6
	EPH	6	6	6	8	1	27
	GenHive	0	0	0	1	0	1
	GISS	0	0	0	0	0	0
	HAEA	1	0	0	0	6	7
	HAHA	0	0	0	0	0	0
	ISEA	0	0	0	0	8	8
	KSATS-HH	0	0	0	0	0	0
	MCHH-S	0	0	0	0	0	0
	ML	1	2	0	4	0	7
	NAHH	0	0	4	0	5	9
	PHunter_{32bit}	6	4	5	3	0	18
	SA-ILS	0	0	0	0	0	0
	SelfSearch	0	0	0	0	3	3
	VNS-TW	6	3	1	0	2	12
	XCJ	0	0	0	0	0	0
Experiments in thesis	PHunter_{64bit}	6	5	10	6	0	27
	SPOT-PHunter	6	10	8	10	0	34

(†: Solutions were available at: http://www.asap.cs.nott.ac.uk/external/chesc2011/raw-results.html.)

Table C.5: Original data of Figure 6.3

Algorithm	n'	r (%)		$\text{Time}_{\text{overall}}$	Time_{L}
		Average	Std dev	(s)	(s)
SPOT (direct)	10	65.759	5.088	8.77	2.76
	20	69.711	3.387	9.28	3.27
	30	71.514	2.822	11.43	5.42
	40	72.409	2.497	17.53	11.52
	50	72.970	2.542	31.87	25.85
SPOT (weighted permutation)	10	68.951	3.022	8.80	2.79
	20	71.052	2.283	9.36	3.35
	30	71.703	2.045	11.45	5.44
	40	72.151	2.020	17.61	11.60
	50	72.367	2.102	31.89	25.88
longest-total-time-first		52.207	—	< 0.01	
NEH		66.636	—	< 0.01	
Avg of random NEHs		66.145	2.181	< 0.01	

Table C.6: Original data of Figure 6.4 (Significant values of p in bold)

Group	Instance	% excess over best-known				Run time (ms)	
		Base LLH	New LLH	F	p	Base LLH	New LLH
No. 5	0	7.043	6.656	170.90	**0.00**	1.76	1.64
versus	1	6.510	6.188	115.81	**0.00**	1.70	1.87
No. 5′	2	6.126	5.701	215.11	**0.00**	1.67	1.59
	3	5.742	5.513	61.40	**0.00**	1.62	1.79
	4	6.149	5.800	151.12	**0.00**	1.51	1.75
	Avg	6.314	5.972		**0.00**	1.65	1.73
No. 6	0	5.589	5.250	176.63	**0.00**	102.95	103.49
versus	1	5.127	4.867	95.50	**0.00**	108.27	104.44
No. 6′	2	4.824	4.475	164.49	**0.00**	101.42	101.35
	3	4.633	4.468	37.20	**0.00**	102.77	103.04
	4	4.810	4.438	226.30	**0.00**	98.36	99.17
	Avg	4.997	4.700		**0.00**	102.75	102.30
No. 9	0	5.651	5.586	6.48	**0.01**	4.82	5.15
versus	1	5.097	5.016	10.24	**0.00**	4.73	4.93
No. 9′	2	4.807	4.750	4.99	**0.03**	4.71	4.71
	3	4.565	4.507	5.20	**0.02**	4.82	4.99
	4	5.043	4.986	4.80	**0.03**	4.65	4.63
	Avg	5.033	4.969		**0.02**	4.75	4.88
No. 10	0	5.730	5.679	3.96	**0.05**	5.12	4.90
versus	1	5.170	5.130	2.65	0.10	5.17	4.76
No. 10′	2	4.878	4.843	1.87	0.17	4.56	4.88
	3	4.621	4.577	3.23	0.07	5.12	4.82
	4	5.113	5.053	5.12	**0.02**	4.46	4.56
	Avg	5.102	5.056		0.08	4.88	4.78

Table C.7: Original data of Table 6.13 (Significant values of p in bold)

	PHunter$_{64bit}$					SPOT-PHunter				
	No. 1	No. 3	No. 8	No. 10	No. 11	No. 1	No. 3	No. 8	No. 10	No. 11
31	6238	26794	6326	11419	26603	6218	26792	6303	11355	26591
runs	6264	26847	6353	11381	26626	6242	26752	6326	11371	26659
	6217	26806	6353	11346	26615	6249	26810	6323	11343	26614
	6248	26822	6350	11358	26602	6226	26755	6326	11333	26616
	6211	26800	6344	11397	26638	6248	26815	6325	11360	26560
	6248	26842	6353	11359	26593	6242	26801	6323	11365	26579
	6226	26827	6353	11350	26600	6242	26776	6350	11359	26659
	6239	26818	6350	11319	26629	6277	26783	6353	11359	26640
	6247	26859	6350	11389	26609	6238	26814	6331	11359	26572
	6229	26783	6366	11375	26603	6247	26811	6350	11368	26596
	6241	26868	6350	11391	26587	6252	26784	6350	11375	26586
	6252	26841	6350	11360	26608	6234	26768	6303	11378	26659
	6265	26770	6335	11365	26616	6259	26783	6329	11340	26585
	6258	26843	6353	11355	26621	6267	26773	6314	11323	26635
	6238	26815	6353	11396	26640	6231	26773	6323	11352	26584
	6256	26760	6363	11402	26683	6228	26784	6310	11393	26659
	6241	26824	6350	11379	26603	6247	26784	6323	11364	26628
	6258	26776	6350	11382	26625	6241	26774	6303	11373	26611
	6248	26778	6350	11415	26610	6213	26790	6303	11359	26586
	6261	26792	6344	11346	26626	6256	26820	6325	11359	26544
	6234	26820	6363	11359	26629	6261	26814	6353	11359	26677
	6250	26858	6325	11359	26622	6233	26830	6323	11359	26567
	6246	26843	6350	11330	26603	6238	26789	6303	11359	26601
	6262	26855	6347	11351	26603	6238	26850	6350	11359	26624
	6226	26868	6353	11362	26601	6248	26817	6309	11373	26634
	6246	26770	6323	11392	26643	6251	26778	6333	11319	26611
	6276	26849	6366	11389	26593	6239	26772	6326	11355	26566
	6238	26852	6353	11389	26608	6238	26814	6318	11381	26604
	6254	26810	6350	11400	26603	6221	26814	6350	11395	26636
	6235	26836	6353	11421	26603	6232	26840	6303	11359	26586
	6215	26811	6353	11370	26643	6242	26825	6303	11329	26577
Min	6211	26760	6323	11319	26587	6213	26752	6303	11319	26544
Med	6246	26822	6350	11375	26609	6242	26790	6323	11359	26604
Max	6276	26868	6366	11421	26683	6277	26850	6353	11395	26677

One-way ANOVA†	No. 1	No. 3	No. 8	No. 10	No. 11
F	0.36	11.47	47.44	7.51	1.21
p	0.55	**0.00**	**0.00**	**0.01**	0.28

†: Factor: PHunter$_{64bit}$ = 1, SPOT-PHunter = 2.

Table C.8: Original data of Figure 6.5

Source	Hyper-heuristic	Score on each test instance					Overall
	(alphabetical)	No. 1	No. 3	No. 8	No. 10	No. 11	score
Results† of CHeSC 2011 (on a 32-bit Java)	ACO-HH	3	0	0	1	0	4
	AdaptHH	10	5	5	9	3	32
	Ant-Q	0	0	0	0	0	0
	AVEG-Nep	0	0	0	0	0	0
	DynILS	0	0	0	0	0	0
	EPH	2	3	4	0	4	13
	GenHive	0	0	1	0	2	3
	GISS	0	0	0	0	0	0
	HAEA	0	0	2	0	1	3
	HAHA	0	0	0	0	0	0
	ISEA	0	0	0	0	0	0
	KSATS-HH	0	0	0	0	0	0
	MCHH-S	0	0	0	0	0	0
	ML	5.5	8	8	3	5	29.5
	NAHH	5.5	8	0	4	0	17.5
	PHunter_{32bit}	0	1.5	0	2	0	3.5
	SA-ILS	0	0	0	0	0	0
	SelfSearch	0	0	0	0	0	0
	VNS-TW	1	4	6	5	10	26
	XCJ	0	0	0	0	0	0
Experiments in thesis	PHunter_{64bit}	4	1.5	3	6	6	20.5
	SPOT-PHunter	8	8	10	9	8	43

(†: Solutions were available at: http://www.asap.cs.nott.ac.uk/external/chesc2011/raw-results.html.)

References

Aaronson, S. (2005, March). Guest column: NP-complete problems and physical reality. *SIGACT News*, *36*(1), 30–52. doi:10.1145/1052796.1052804

Aha, D. W., Kibler, D., & Albert, M. K. (1991, January). Instance-based learning algorithms. *Machine Learning*, *6*(1), 37–66. doi:10.1023/A:1022689900470

Ahuja, R. K., Ergun, O., Orlin, J. B., & Punnen, A. P. (2002). A survey of very large-scale neighborhood search techniques. *Discrete Applied Mathematics*, *123*(1-3), 75–102. doi:10.1016/S0166-218X(01)00338-9

Aickelin, U., Burke, E. K., & Li, J. (2009, April). An evolutionary squeaky wheel optimization approach to personnel scheduling. *IEEE Transactions on Evolutionary Computation*, *13*(2), 433–443. doi:10.1109/TEVC.2008.2004262

Allahverdi, A., & Al-Anzi, F. S. (2002, July). Using two-machine flowshop with maximum lateness objective to model multimedia data objects scheduling problem for WWW applications. *Comput. Oper. Res. 29*(8), 971–994. doi:10.1016/S0305-0548(00)00097-6

Applegate, D. L., Bixby, R. E., Chvátal, V., & Cook, W. J. (2006). *The traveling salesman problem: a computational study*. Princeton Series in Applied Mathematics. Princeton, NJ, USA: Princeton University Press.

Applegate, D. L., Bixby, R. E., Chvátal, V., Cook, W. J., Espinoza, D. G., Goycoolea, M., & Helsgaun, K. (2009). Certification of an optimal TSP tour through 85,900 cities. *Operations Research Letters*, *37*(1), 11–15. doi:10.1016/j.orl.2008.09.006

Applegate, D. L., Bixby, R. E., Chvátal, V., & Cook, W. J. (2001). TSP cuts which do not conform to the template paradigm. In M. Jünger & D. Naddef (Eds.), *Computational combinatorial optimization* (Vol. 2241, pp. 261–304). Lecture Notes in Computer Science. London, UK: Springer-Verlag. doi:10.1007/3-540-45586-8_7

Ausiello, G., Protasi, M., Marchetti-Spaccamela, A., Gambosi, G., Crescenzi, P., & Kann, V. (1999). *Complexity and approximation: combinatorial optimization problems and their approximability properties* (1st). Secaucus, NJ, USA: Springer-Verlag New York, Inc. doi:10.1007/978-3-642-58412-1

Babb, J., & Currie, J. (2008, July). The brachistochrone problem: mathematics for a broad audience via a large context problem. *The Montana Mathematics Enthusiast*, *5*(3), 169–184. Retrieved April 4, 2012, from http://www.math.umt.edu/tmme/vol5no2and3/TMME_vol5nos2and3_a1_pp.169_184.pdf

Bader El Den, M., & Poli, R. (2009, May). Grammar-based genetic programming for timetabling. In *2009 IEEE congress on evolutionary computation (cec'09)* (pp. 2532–2539). doi:10.1109/CEC.2009.4983259

Barnhart, C., Johnson, E. L., Nemhauser, G. L., Savelsbergh, M. W. P., & Vance, P. H. (1998, March). Branch-and-price: column generation for solving huge integer programs. *Operations Research*, *46*(3), 316–329. doi:10.1287/opre.46.3.316

Bentley, J. J. (1992). Fast algorithms for geometric traveling salesman problems. *OSAR Journal on Computing*, *4*, 387–411. doi:10.1287/ijoc.4.4.387

Bernoulli, J. (1696, June). Problema novum ad cuius solutionem Mathematici invitantur. *Acta Eruditorum*, *15*, 264–269.

Bernstein, M. (1987, January). Finding heuristics for flowshop scheduling. *SIGART Newsletter*, (99), 32–33. doi:10.1145/24667.24670

Birattari, M. (2009). *Tuning metaheuristics: a machine learning perspective* (J. Kacprzyk, Ed.). Studies in Computational Intelligence. Berlin, Germany: Springer-Verlag. doi:10.1007/978-3-642-00483-4

Blum, A. L., & Rivest, R. L. (1992). Training a 3-node neural network is NP-complete. *Neural Networks*, *5*(1), 117–127. doi:10.1016/S0893-6080(05)80010-3

Blum, C., & Roli, A. (2003, September). Metaheuristics in combinatorial optimization: overview and conceptual comparison. *ACM Computer Survey*, *35*(3), 268–308. doi:10.1145/937503.937505

Bonabeau, E., Dorigo, M., & Theraulaz, G. (1999). *Swarm intelligence: from natural to artificial systems*. Santa Fe Institute Studies in the Sciences of Complexity. New York, NY, USA: Oxford University Press. Retrieved April 4, 2012, from http://books.google.com.hk/books?id=PvTDhzqMr7cC

Bose, I., & Mahapatra, R. K. (2001). Business data mining — a machine learning perspective. *Information & Management*, *39*(3), 211–225. doi:10.1016/S0378-7206(01)00091-X

Boyan, J., & Moore, A. (2000). ‘‘STAGE’’ learning for local search. *Neural computing surveys*, *3*, 35–38. Retrieved April 4, 2012, from http://ftp.icsi.berkeley.edu/ftp/pub/ai/jagota/vol3_1.pdf

Boyan, J. (1998, August). *Learning evaluation functions for global optimization*. (Doctoral dissertation, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, USA). Retrieved April 4, 2012, from http://www.ri.cmu.edu/publication_view.html?pub_id=2954

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1983). *Classification and regression trees*. Wadsworth Statistics / Probability series. Belmont, CA, USA: Wadsworth Publishing Co. Inc.

Brown, D. E., Pittard, C. L., & Park, H. (1996). Classification trees with optimal multivariate decision nodes. *Pattern Recognition Letters*, *17*(7), 699–703. doi:10.1016/0167-8655(96)00033-5

Burke, E. K., Hyde, M., & Kendall, G. (2006). Evolving bin packing heuristics with genetic programming. In *Proceedings of the 9th international conference on parallel problem solving from nature (ppsn'06)* (Vol. 4193, pp. 860–869). Lecture Notes in Computer Science. Germany: Springer-Verlag. doi:10.1007/11844297_87

Burke, E. K., Hyde, M., Kendall, G., Ochoa, G., Ozcan, E., & Qu, R. (2010). *Hyper-heuristics: a survey of the state of the art* (tech. rep. No. NOTTCS-TR-SUB-0906241418-2747). School of Computer Science and Information Technology, University of Nottingham. Nottingham NG8 1BB, UK. Retrieved April 4, 2012, from http://www.cs.nott.ac.uk/~gxo/papers/hhsurvey.pdf

Burke, E. K., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., & Woodward, J. R. (2010). A classification of hyper-heuristic approaches. In M. Gendreau & J.-Y. Potvin (Eds.), *Handbook of metaheuristics* (Vol. 146, pp. 449–468). International Series in Operations Research & Management Science. Springer US. doi:10.1007/978-1-4419-1665-5_15

Burke, E. K., Hyde, M., Kendall, G., & Woodward, J. (2007). Automatic heuristic generation with genetic programming: evolving a jack-of-all-trades or a master of one. In *Proceedings of the 9th annual conference on genetic and evolutionary computation (gecco'07)* (pp. 1559–1565). New York, NY, USA: ACM. doi:10.1145/1276958.1277273

Burke, E. K., Kendall, G., Newall, J., Hart, E., Ross, P., & Schulenburg, S. (2003). Hyper-heuristics: an emerging direction in modern search technology. In F. Glover & G. Kochenberger (Eds.), *Handbook of metaheuristics* (Vol. 57, pp. 457–474). International Series in Operations Research & Management Science. Springer New York. doi:10.1007/0-306-48056-5_16

Burke, E. K., Petrovic, S., & Qu, R. (2006, April). Case-based heuristic selection for timetabling problems. *J. of Scheduling*, *9*(2), 115–132. doi:10.1007/s10951-006-6775-y

Campbell, H. G., Dudek, R. A., & Smith, M. L. (1970). A heuristic algorithm for the *n* job, *m* machine sequencing problem. *Management Science*, *16*(10), B630–B637. doi:doi:10.1287/mnsc.16.10.B630

Ceberio, J., Irurozki, E., Mendiburu, A., & Lozano, J. A. (2012). A review on estimation of distribution algorithms in permutation-based combinatorial optimization problems. *Progress in Artificial Intelligence*, *1*(1), 103–117. doi:10.1007/s13748-011-0005-3

Černý, V. (1985, January). Thermodynamical approach to the traveling salesman problem: an efficient simulation algorithm. *Journal of Optimization Theory and Applications*, *45*(1), 41–51. doi:10.1007/BF00940812

Chan, C. Y., Xue, F., Ip, W. H., & Cheung, C. F. (2012, January). A hyper-heuristic inspired by pearl hunting. In Y. Hamadi & M. Schoenauer (Eds.), *Learning and intelligent optimization* (pp. 349–353). Lecture Notes

in Computer Science. Springer-Verlag. doi:10.1007/978-3-642-34413-8_26

Clark, P., & Niblett, T. (1989). The CN2 induction algorithm. *Machine Learning*, *3*, 261–283. doi:10.1007/BF00116835

Cohen, W. W. (1995). Fast effective rule induction. In *Proceedings of the twelfth international conference on machine learning (ml95)* (pp. 115–123). Tahoe City, CA, USA. Retrieved April 4, 2012, from http://www.cs.cmu.edu/~wcohen/postscript/ml-95-ripper.ps

Colorni, A., Dorigo, M., & Maniezzo, V. (1992). Distributed optimization by ant colonies. In F. J. Varela & P. Bourgine (Eds.), *Towards a practice of autonomous systems: proceedings of the first european conference on artificial life* (pp. 134–142). Cambridge, MA, USA: MIT Press.

Cook, W. J., & Seymour, P. (2003, July). Tour merging via branch-decomposition. *INFORMS Journal on Computing*, *15*(3), 233–248. doi:10.1287/ijoc.15.3.233.16078

Cotta, C., & Troya, J. M. (2003, April). Embedding branch and bound within evolutionary algorithms. *Applied Intelligence*, *18*(2), 137–153. doi:10.1023/A:1021934325079

Croes, G. A. (1958, December). A method for solving traveling-salesman problems. *Operations Research*, *6*(6), 791–812. doi:10.1287/opre.6.6.791

Crowston, W. B., Glover, F., Thompson, G. L., & Trawick, J. D. (1963). *Probabilistic and parametric learning combinations of local job shop scheduling rules*. ONR Research Memorandum. Pittsburgh, PA, USA: Defense Technical Information Center.

Cruz-Reyes, L., Gómez-Santillán, C., Pérez-Ortega, J., Landero, V., Quiroz, M., & Ochoa, A. (2012, March). Algorithm selection: from meta-learning to

hyper-heuristics. In V. M. Koleshko (Ed.), *Intelligent systems* (Chap. 4, pp. 77–102). Rijeka, Croatia: InTech. doi:10.5772/2350

Culberson, J. C. (1998, June). On the futility of blind search: an algorithmic view of "no free lunch". *Evolutionary Computation*, *6*(2), 109–127. doi:10.1162/evco.1998.6.2.109

Dannenbring, D. G. (1977, July). An evaluation of flow shop sequencing heuristics. *Management Science*, *23*(11), 1174–1182. Retrieved from http://www.jstor.org/stable/2630656

Davis, M. (1965). *The undecidable: basic papers on undecidable propositions, unsolvable problems and computable functions*. Hewlett, New York, NY, USA: Raven Press.

Denzinger, J., Fuchs, M., & Fuchs, M. (1997). High performance ATP systems by combining several AI methods. In *Proceedings of the 15th international joint conference on artifical intelligence — volume 1* (pp. 102–107). IJCAI'97. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. Retrieved April 4, 2012, from http://pages.cpsc.ucalgary.ca/~denzinge/papers/Denzinger.SR-96-09.ps.gz

Denzinger, J., & Offermann, T. (1999). On cooperation between evolutionary algorithms and other search paradigms. In *Proceedings of the 1999 congress on evolutionary computation (cec'99)* (pp. 2317–2324). doi:10.1109/CEC.1999.785563

de Werra, D., & Hertz, A. (1989). Tabu-search techniques: a tutorial and an application to neural networks. *OR Spektrum*, *11*(3), 131–141. doi:10.1007/BF01720782

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, *1*, 269–271. doi:10.1007/BF01386390

Dorigo, M., & Gambardella, L. M. (1997a). Ant colonies for the travelling salesman problem. *Biosystems*, *43*(2), 73–81. doi:10.1016/S0303-2647(97)01708-5

Dorigo, M., & Gambardella, L. M. (1997b). Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, *1*(1), 53–66. doi:10.1109/4235.585892

Dorigo, M., Maniezzo, V., & Colorni, A. (1991). *Positive feedback as a search strategy* (tech. rep. No. 91-016). Dipartimento di Elettronica, Politecnico di Milano. Milan, Italy. Retrieved April 4, 2012, from http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.52.6342

Eiben, A. E., Hinterding, R., & Michalewicz, Z. (1999, July). Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, *3*(2), 124–141. doi:10.1109/4235.771166

Eiben, A. E., & Smith, J. E. (2003). *Introduction to evolutionary computing*. Natural Computing Series. Berlin, Germany: Springer-Verlag. Retrieved April 4, 2012, from http://www.cs.vu.nl/~gusz/ecbook/ecbook.html

Fan, W., & Xue, F. (2006). Optimize cooperative agents with organization in distributed scheduling system. In K. L. De-Shuang Huang & G. W. Irwin (Eds.), *Computational intelligence* (Vol. 4114, pp. 502–509). Lecture Notes in Artificial Intelligence. Germany: Springer-Verlag. doi:10.1007/978-3-540-37275-2_61

Feo, T. A., & Resende, M. G. C. (1989). A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, *8*(2), 67–71. doi:10.1016/0167-6377(89)90002-3

Feo, T. A., & Resende, M. G. C. (1995). Greedy randomized adaptive search procedures. *Journal of Global Optimization*, *6*, 109–133. doi:10.1007/BF01096763

Fisher, R. A. (1936). The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, *7*(2), 179–188. doi:10.1111/j.1469-1809.1936.tb02137.x

French, A. P., Robinson, A. C., & Wilson, J. M. (2001, November). Using a hybrid genetic-algorithm/branch and bound approach to solve feasibility and optimization integer programming problems. *Journal of Heuristics*, *7*(6), 551–564. doi:10.1023/A:1011921025322

Fukunaga, A. S. (2008, March). Automated discovery of local search heuristics for satisfiability testing. *Evolutionary Computation*, *16*(1), 31–61. doi:10.1162/evco.2008.16.1.31

Gama, J. (1999). Discriminant trees. In *Proceedings of the sixteenth international conference on machine learning (icml'99)* (pp. 134–142). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. Retrieved April 4, 2012, from http://www.liaad.up.pt/~jgama/ml99.ps.gz

Gambardella, L. M., & Dorigo, M. (1995). Ant-Q: a reinforcement learning approach to the traveling salesman problem. In A. Prieditis & S. Russell (Eds.), *Machine learning: proceedings of the twelfth international conference on machine learning* (pp. 252–260). San Francisco, CA: Morgan Kaufmann Publishers. Retrieved April 4, 2012, from http://www.idsia.ch/~luca/ml95.pdf

Garey, M. R., & Johnson, D. S. (1979). *Computers and intractability; a guide to the theory of NP-completeness*. New York, NY, USA: W. H. Freeman & Co. doi:10.1137/1024022

Garey, M. R., Johnson, D. S., & Sethi, R. (1976, May). The complexity of flowshop and jobshop scheduling. *Mathematics of operations research*, *1*(2), 117–129. doi:10.1287/moor.1.2.117

Gendreau, M., & Potvin, J.-Y. (2010). *Handbook of metaheuristics* (2nd) (F. S. Hillie, Ed.). International Series in Operations Research & Management Science. New York, NY, USA: Springer Science+Business Media. doi:10.1007/978-1-4419-1665-5

Gerdts, M. (2012). *Optimal control of ODEs and DAEs*. Göttingen, Germany: Walter De Gruyter Inc. doi:10.1515/9783110249996

Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, *13*(5), 533–549. doi:10.1016/0305-0548(86)90048-1

Glover, F., & Laguna, M. (1997). *Tabu search*. Kluwer Academic Publishers. doi:10.1007/978-1-4615-6089-0

Gomes, C. P., & Selman, B. (1997). Algorithm portfolio design: theory vs. practice. In *Proceedings of the thirteenth conference on uncertainty in artificial intelligence* (pp. 190–197). UAI'97. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. Retrieved April 4, 2012, from http://dl.acm.org/citation.cfm?id=2074226.2074249

Grefenstette, J. J. (1986, January). Optimization of control parameters for genetic algorithms. *IEEE Transactions on Systems, Man and Cybernetics*, *16*(1), 122–128. doi:10.1109/TSMC.1986.289288

Gupta, J. N. D. (1975). A search algorithm for the generalized flowshop scheduling problem. *Computers & Operations Research*, *2*(2), 83–90. doi:10.1016/0305-0548(75)90011-8

Gutin, G., & Glover, F. (2005). Further extension of the TSP assign neighborhood. *Journal of Heuristics*, *11*(5-6), 501–505. doi:10.1007/s10732-005-3487-y

Hackley, C. W. (1847). *Elementary course of geometry*. New York, NY, USA: Harper & Brothers, Publishers. Retrieved April 4, 2012, from http://www.google.com.hk/books?id=jWILAAAAYAAJ

Hall, M. A. (1999, April). *Correlation-based feature selection for machine learning*. (Doctoral dissertation, Department of Computer Science, University of Waikato, Hamilton, NewZealand). Retrieved April 4, 2012, from http://www.cs.waikato.ac.nz/~mhall/thesis.pdf

Held, M., & Karp, R. M. (1970, December). The traveling-salesman problem and minimum spanning trees. *Operations Research*, *18*(6), 1138–1162. doi:10.1287/opre.18.6.1138

Heller, J. (1960, April). Some numerical experiments for an M x J flow shop and its decision-theoretical aspects. *Operations Research*, *8*(2), 178–184. doi:10.1287/opre.8.2.178

Helsgaun, K. (2000). An effective implementation of the Lin-Kernighan traveling salesman heuristic. *European Journal of Operational Research*, *126*(1), 106–130. doi:10.1016/S0377-2217(99)00284-2

Helsgaun, K. (2009). General *k*-opt submoves for the Lin-Kernighan TSP heuristic. *Mathematical Programming Computation*, *1*, 119–163. doi:10.1007/s12532-009-0004-6

Hilario, M., Kalousis, A., Nguyen, P., & Woznica, A. (2009, September). A data mining ontology for algorithm selection and meta-mining. In V. Podpečan, N. Lavrač, J. N. Kok & J. de Bruin (Eds.), *Proceedings of the 2009 european conference on machine learning and principles and practice*

of knowledge discovery in databases (ecml pkdd 2009) (pp. 76–87). Bled, Slovenia. Retrieved April 4, 2012, from http://www.ecmlpkdd2009.net/wp-content/uploads/2008/09/service-oriented-knowledge-discovery_2.pdf#page=82

Holland, J. H. (1992). *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control and artificial intelligence*. Cambridge, MA, USA: MIT Press.

Ho, N. B., & Tay, J. C. (2005, September). Evolving dispatching rules for solving the flexible job-shop problem. In *The 2005 IEEE congress on evolutionary computation* (Vol. 3, pp. 2848–2855). doi:10.1109/CEC.2005.1555052

Hsiao, P.-C., Chiang, T.-C., & Fu, C. (2011, September). A variable neighborhood search-based hyperheuristic for corss-domain optimization problems in CHeSC 2011 competition. (p. 89). The OR Society. Nottingham, UK.

Huberman, B. A., Lukose, R. M., & Hogg, T. (1997, January). An economics approach to hard computational problems. *Science*, *275*(5296), 51–54. doi:10.1126/science.275.5296.51

Hunt, J. E., & Cooke, D. E. (1996, April). Learning using an artificial immune system. *Journal of Network and Computer Applications*, *19*(2), 189–212. doi:10.1006/jnca.1996.0014

Hutchinson, J. E. (1981). Fractals and self-similarity. *Indiana University Mathematics Journal*, *30*(5), 713–747. Retrieved April 4, 2012, from http://maths.anu.edu.au/~john/Assets/Research%20Papers/fractals_self-similarity.pdf

Hutter, F., Hoos, H. H., Leyton-Brown, K., & Stützle, T. (2009, September). ParamILS: an automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, *36*(1), 267–306. Retrieved April 4, 2012, from http://www.jair.org/media/2861/live-2861-4703-jair.pdf

Ibaraki, T., & Muroga, S. (1970, January). Adaptive linear classifier by linear programming. *IEEE Transactions on Systems Science and Cybernetics*, *6*(1), 53–62. doi:10.1109/TSSC.1970.300329

Ignall, E., & Schrage, L. (1965, June). Application of the Branch and Bound technique to some flow-shop scheduling problems. *Operations Research*, *13*(3), 400–412. doi:10.1287/opre.13.3.400

Jakobović, D., Jelenković, L., & Budin, L. (2007). Genetic programming heuristics for multiple machine scheduling. In *Proceedings of the 10th european conference on genetic programming (eurogp'07)* (Vol. 4445, pp. 321–330). Lecture Notes in Computer Science. Germany: Springer-Verlag. doi:10.1007/978-3-540-71605-1_30

John, G. H., & Langley, P. (1995). Estimating continuous distributions in Bayesian classifiers. In *Proceedings of the eleventh conference on uncertainty in artificial intelligence (uai'95)* (pp. 338–345). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. Retrieved April 4, 2012, from http://dl.acm.org/citation.cfm?id=2074158.2074196

Johnson, D. S., & McGeoch, L. A. (1997). Traveling salesman problem: a case study in local optimization. In E. H. L. Aarts & J. K. Lenstra (Eds.), *Local search in combinatorial optimization* (Chap. 8, pp. 215–310). London, UK: John Wiley and Sons. Retrieved April 4, 2012, from http://www2.research.att.com/~dsj/papers/TSPchapter.pdf

Johnson, D. S., & McGeoch, L. A. (2002). Experimental analysis of heuristics for STSP. In G. Gutin & A. P. Punnen (Eds.), *The traveling salesman problem and its variations* (Chap. 9, pp. 369–443). New York, NY, USA: Kluwer Academic Publishers. doi:10.1007/b101971

Johnson, S. M. (1954). Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly*, *1*(1), 61–68. doi:10.1002/nav.3800010110

Joslin, D. E., & Clements, D. P. (1999, May). ''squeaky wheel" optimization. *Journal of Artificial Intelligence Research*, *10*(1), 353–373. Retrieved April 4, 2012, from http://www.jair.org/media/561/live-561-1759-jair.pdf

Kadioglu, S., Malitsky, Y., Sellmann, M., & Tierney, K. (2010). Isac – instance-specific algorithm configuration. In *Proceedings of 19th European conference on artificial intelligence (ecai 2010)* (pp. 751–756). Amsterdam, Netherlands: IOS Press. Retrieved April 4, 2012, from http://www.itu.dk/people/kevt/papers/isac-ecai2010.pdf

Kanda, J., Carvalho, A., Hruschka, E., & Soares, C. (2011, August). Selection of algorithms to solve traveling salesman problems using meta-learning. *International Journal of Hybrid Intelligent Systems*, *8*(3), 117–128. doi:10.3233/HIS-2011-0133

Karp, R. M. (1972). Reducibility among combinatorial problems. In R. E. Miller & J. W. Thatcher (Eds.), *Complexity of computer computations* (pp. 85–103). New York, NY, USA: Plenum Press. doi:10.1007/978-1-4684-2001-2_9

Kassim, H. O., & Cadbury, R. G. (1996). The place of the computer in chemical engineering education. *Computers & Chemical Engineering*, *20, Sup-*

plement 2, S1341–S1346. European Symposium on Computer Aided Process Engineering 6. doi:10.1016/0098-1354(96)00230-X

Keller, R. E., & Poli, R. (2007, September). Linear genetic programming of parsimonious metaheuristics. In *2007 IEEE congress on evolutionary computation (cec 2007)* (pp. 4508–4515). doi:10.1109/CEC.2007.4425062

Keller, R. E., & Poli, R. (2008). Cost-benefit investigation of a genetic-programming hyperheuristic. In *Proceedings of the evolution artificielle, 8th international conference on artificial evolution (ea'07)* (Vol. 4926, pp. 13–24). Lecture Notes in Computer Science. Germany: Springer-Verlag. doi:10.1007/978-3-540-79305-2_2

Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization. In *Proceedings of IEEE international conference on neural networks* (Vol. 4, pp. 1942–1948). IEEE Press. doi:10.1109/ICNN.1995.488968

Kennedy, J., Eberhart, R., & Shi, Y. (2001). *Swarm intelligence*. San Diego, CA: Morgan Kaufmann Publishers.

Kim, B.-I., & Wy, J. (2010). Last two fit augmentation to the well-known construction heuristics for one-dimensional bin-packing problem: an empirical study. *The International Journal of Advanced Manufacturing Technology*, *50*, 1145–1152. doi:10.1007/s00170-010-2572-z

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983, May). Optimization by simulated annealing. *Science*, *220*(4598), 671–680. doi:10.1126/science.220.4598.671

Kleunen, J. P. C. (1976). Computers and operations research: a survey. *Computers & Operations Research*, *3*(4), 327–335. doi:10.1016/0305-0548(76)90015-0

Knuth, D. E., & Moore, R. W. (1975). An analysis of alpha-beta pruning. *Artificial Intelligence*, *6*(4), 293–326. doi:10.1016/0004-3702(75)90019-3

Ko, K.-I., & Friedman, H. (1982). Computational complexity of real functions. *Theoretical Computer Science*, *20*(3), 323–352. doi:10.1016/S0304-3975(82)80003-0

Koonce, D. A., & Tsai, S.-C. (2000). Using data mining to find patterns in genetic algorithm solutions to a job shop schedule. *Comput. Ind. Eng. 38*(3), 361–374. doi:10.1016/S0360-8352(00)00050-4

Kotsiantis, S. B. (2007, October). Supervised machine learning: a review of classification techniques. *Informatica*, *31*(3), 249–268. Retrieved April 4, 2012, from http://www.informatica.si/vol31.htm#No3

Kotsiantis, S. B., Zaharakis, I. D., & Pintelas, P. E. (2006, November). Machine learning: a review of classification and combining techniques. *Artificial Intelligence Review*, *26*(3), 159–190. doi:10.1007/s10462-007-9052-3

Koza, J. R. (1992). *Genetic programming: on the programming of computers by means of natural selection*. Cambridge, MA, USA: MIT Press.

Kriegel, H.-P., Kröger, P., Schubert, E., & Zimek, A. (2008). A general framework for increasing the robustness of PCA-based correlation clustering algorithms. In *Proceedings of the 20th international conference on scientific and statistical database management (ssdbm '08)* (Vol. 5069, pp. 418–435). Lecture Notes in Computer Science. Germany: Springer-Verlag. doi:10.1007/978-3-540-69497-7_27

Kwak, C., & Yih, Y. (2004, February). Data-mining approach to production control in the computer-integrated testing cell. *IEEE Transactions on Robotics And Automation*, *20*(1), 107–116. doi:10.1109/TRA.2003.819595

Lachenbruch, P. A., & Goldstein, M. (1979). Discriminant analysis. *Biometrics*, *35*(1), 69–85. doi:10.2307/2529937

Lawler, E. L., & Wood, D. E. (1966, August). Branch-and-bound methods: a survey. *Operations Research*, *14*(4), 699–719. doi:10.1287/opre.14.4.699

Le Cessie, S., & Van Houwelinge, J. C. (1992). Ridge estimators in logistic regression. *Applied Statistics*, *41*(1), 191–201. doi:10.2307/2347628

Lee, C.-Y., Piramuthu, S., & Tsai, Y.-K. (1997). Job shop scheduling with a genetic algorithm and machine learning. *International Journal of Production Research*, *35*(4), 1171–1191. doi:10.1080/002075497195605

Leyton-Brown, K., Nudelman, E., Andrew, G., McFadden, J., & Shoham, Y. (2003). A portfolio approach to algorithm select. In *Proceedings of the 18th international joint conference on artificial intelligence* (pp. 1542–1543). IJCAI'03. Acapulco, Mexico: Morgan Kaufmann Publishers Inc. Retrieved April 4, 2012, from http://robotics.stanford.edu/~eugnud/papers/portfolio-IJCAI.pdf

Li, L., Ju, S., & Zhang, Y. (2008, October). Improved ant colony optimization for the traveling salesman problem. In *2008 international conference on intelligent computation technology and automation (icicta)* (pp. 76–80). doi:10.1109/ICICTA.2008.265

Lindner, G., & Studer, R. (1999). AST: support for algorithm selection with a CBR approach. In J. Zytkow & J. Rauch (Eds.), *Principles of data mining and knowledge discovery* (Vol. 1704, pp. 418–423). Lecture Notes in Computer Science. Germany: Springer. doi:10.1007/978-3-540-48247-5_52

Lin, S. (1965, December). Computer solutions of the traveling salesman problem. *Bell System Technical Journal*, *44*(10), 2245–2269. Retrieved

April 4, 2012, from http://www.alcatel-lucent.com/bstj/vol44-1965/articles/bstj44-10-2245.pdf

Lin, S., & Kernighan, B. W. (1973). An effective heuristic algorithm for the travelling-salesman problem. *Operations Research*, *21*, 498–516. doi:10.1287/opre.21.2.498

Liu, H., & Motoda, H. (2001). *Instance selection and construction for data mining*. Norwell, MA, USA: Kluwer Academic Publishers.

Li, X., & Olafsson, S. (2005). Discovering dispatching rules using data mining. *Journal of Scheduling*, *8*(6), 515–527. doi:10.1007/s10951-005-4781-0

Lobjois, L., & Lemaître, M. (1998). Branch and bound algorithm selection by performance prediction. In *Proceedings of the fifteenth national/tenth conference on artificial intelligence/innovative applications of artificial intelligence* (pp. 353–358). AAAI '98/IAAI '98. Menlo Park, CA, USA: American Association for Artificial Intelligence. Retrieved April 4, 2012, from http://aaaipress.org/Papers/AAAI/1998/AAAI98-050.pdf

Lobo, F. G., & Goldberg, D. E. (2004, December). The parameter-less genetic algorithm in practice. *Inf. Sci. Inf. Comput. Sci. 167*(1-4), 217–232. doi:10.1016/j.ins.2003.03.029

Lourenço, H., Martin, O., & Stützle, T. (2003). Iterated local search. In F. Glover & G. Kochenberger (Eds.), *Handbook of metaheuristics* (Chap. 11, Vol. 57, pp. 320–353). International Series in Operations Research & Management Science. New York, NY, USA: Springer New York. doi:10.1007/0-306-48056-5_11

Mandelbrot, B. (1967, May). How long is the coast of Britain? statistical self-similarity and fractional dimension. *Science*, *156*(3775), 636–638. doi:10.1126/science.156.3775.636

Marinakis, Y., Migdalas, A., & Pardalos, P. M. (2005a). A hybrid genetic-GRASP algorithm using Lagrangean relaxation for the traveling salesman problem. *Journal of Combinatorial Optimization*, *10*, 311–326. doi:10.1007/s10878-005-4921-7

Marinakis, Y., Migdalas, A., & Pardalos, P. M. (2005b, December). Expanding neighborhood GRASP for the traveling salesman problem. *Computational Optimization and Applications*, *32*(3), 231–257. doi:10.1007/s10589-005-4798-5

Markovitch, S., & Rosenstein, D. (2002, October). Feature generation using general constructor functions. *Machine Learning*, *49*(1), 59–98. doi:10.1023/A:1014046307775

McCallum, A., & Nigam, K. (1998). A comparison of event models for Naive Bayes text classification. In *Proceedings of aaai-98 workshop on learning for text categorization* (pp. 41–48). Madison, WI, USA: AAAI Press. Retrieved April 4, 2012, from http://www.aaai.org/Papers/Workshops/1998/WS-98-05/WS98-05-007.pdf

Merrill, A. S. (1919, January). An isoperimetric problem with variable endpoints. *American Journal of Mathematics*, *41*(1), 60–78. doi:10.2307/2370478

Miller, D. L., & Pekny, J. F. (1995). A staged primal-dual algorithm for perfect *b*-matching with edge capacities. *ORSA Journal on Computing*, *7*(3), 298–320. doi:10.1287/ijoc.7.3.298

Milor, L., & Sangiovanni-Vincentelli, A. L. (1994, June). Minimizing production test time to detect faults in analog circuits. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, *13*(6), 796–813. doi:10.1109/43.285252

Mısır, M., Verbeeck, K., Causmaecker, P. D., & Vanden Berghe, G. (2012). An intelligent hyper-heuristic framework for CHeSC 2011. In *Sixth international conference on learning and intelligent optimization (lion 6)*. Lecture Notes in Computer Science. Springer.

Mladenović, N., & Hansen, P. (1997). Variable neighborhood search. *Computers & Operations Research*, *24*(11), 1097–1100. doi:10.1016/S0305-0548(97)00031-2

Moscato, P., & Cotta, C. (2003). A gentle introduction to memetic algorithms. In F. Glover & G. Kochenberger (Eds.), *Handbook of metaheuristics* (Vol. 57, pp. 105–144). International Series in Operations Research & Management Science. New York, NY, USA: Springer New York. doi:10.1007/0-306-48056-5_5

Mühlenbein, H., Bendisch, J., & Voigt, H.-M. (1996). From recombination of genes to the estimation of distributions II. continuous parameters. In *PPSN IV: proceedings of the 4th international conference on parallel problem solving from nature* (pp. 188–197). London, UK: Springer-Verlag. doi:10.1007/3-540-61723-X_983

Mühlenbein, H., & Mahnig, T. (1999, December). Fda — a scalable evolutionary algorithm for the optimization of additively decomposed functions. *Evolutionary Computation*, *7*(4), 353–376. doi:10.1162/evco.1999.7.4.353

Mühlenbein, H., & Paaß, G. (1996). From recombination of genes to the estimation of distributions I. binary parameters. In *PPSN IV: proceedings of the 4th international conference on parallel problem solving from nature* (pp. 178–187). London, UK: Springer-Verlag. doi:10.1007/3-540-61723-X_982

Murthy, S. K. (1998). Automatic construction of decision trees from data: a multi-disciplinary survey. *Data Mining and Knowledge Discovery*, *2*, 345–389. doi:10.1023/A:1009744630224

Nawaz, M., Enscore, E. E., & Ham, I. (1983). A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega*, *11*(1), 91–95. doi:10.1016/0305-0483(83)90088-9

Nemhauser, G. L., & Wolsey, L. A. (1988). *Integer and combinatorial optimization*. New York, NY, USA: Wiley-Interscience.

Ochoa, G., Hyde, M., Curtois, T., Vázquez-Rodríguez, J. A., Walker, J., Gendreau, M., … Burke, E. K. (2012). HyFlex: a benchmark framework for cross-domain heuristic search. LNCS, *7245*, 136–147. doi:10.1007/978-3-642-29124-1_12

Ochoa, G., Vázquez-Rodríguez, J. A., Petrovic, S., & Burke, E. K. (2009). Dispatching rules for production scheduling: a hyper-heuristic landscape analysis. In *Proceedings of the eleventh conference on congress on evolutionary computation (cec'09)* (pp. 1873–1880). Piscataway, NJ, USA: IEEE Press. doi:10.1109/CEC.2009.4983169

Olafsson, S., & Li, X. (2010). Learning effective new single machine dispatching rules from optimal scheduling data. *International Journal of Production Economics*, *128*(1), 118–126. doi:10.1016/j.ijpe.2010.06.004

Palmer, D. S. (1965). Sequencing jobs through a multi-stage process in the minimum total time – a quick method of obtaining a near optimum. *OR*, *16*(1), 101–107. doi:doi:10.2307/3006688

Papadimitriou, C. H. (1977). The Euclidean travelling salesman problem is NP-complete. *Theoretical Computer Science*, *4*(3), 237–244. doi:10.1016/0304-3975(77)90012-3

Papadimitriou, C. H., & Steiglitz, K. (1982). *Combinatorial optimization: algorithms and complexity*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc.

Papadimitriou, C. H., & Vempala, S. (2006, February). On the approximability of the traveling salesman problem. *Combinatorica*, *26*(1), 101–120. doi:10.1007/s00493-006-0008-z

Pelikan, M., Goldberg, D. E., & Cantú-Paz, E. (1999). BOA: the Bayesian optimization algorithm. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela & R. E. Smith (Eds.), *Proceedings of the genetic and evolutionary computation conference (gecco-99)* (Vol. 1, pp. 525–532). Orlando, FL, USA: Morgan Kaufmann Publishers. Retrieved April 4, 2012, from http://archive.org/details/BoaTheBayesianOptimizationAlgorithm

Petersen, L., Minkkinen, P., & Esbensen, K. H. (2005). Representative sampling for reliable data analysis: theory of sampling. *Chemometrics and Intelligent Laboratory Systems*, *77*(1-2), 261–277. doi:10.1016/j.chemolab.2004.09.013

Pillay, N., & Banzhaf, W. (2007). A genetic programming approach to the generation of hyper-heuristics for the uncapacitated examination timetabling problem. In *Proceedings of the aritficial intelligence 13th portuguese conference on progress in artificial intelligence (epia'07)* (Vol. 4874, pp. 223–234). Lecture Notes in Computer Science. Germany: Springer-Verlag. doi:10.1007/978-3-540-77002-2_19

Poli, R., & Graff, M. (2009). There is a free lunch for hyper-heuristics, genetic programming and computer scientists. In *Proceedings of the 12th European conference on genetic programming (eurogp '09)* (Vol. 5481, pp. 195–

207). Lecture Notes in Computer Science. Germany: Springer. doi:10.1007/978-3-642-01181-8_17

Puchinger, J., & Raidl, G. (2005). Combining metaheuristics and exact algorithms in combinatorial optimization: a survey and classification. In J. Mira & J. Álvarez (Eds.), *Artificial intelligence and knowledge engineering applications: a bioinspired approach* (Vol. 3562, pp. 113–124). Lecture Notes in Computer Science. Germany: Springer. doi:10.1007/11499305_5

Punnen, A. P. (2004). The traveling salesman problem: applications, formulations and variations. In G. Gutin & A. P. Punnen (Eds.), *The traveling salesman problem and its variations* (Chap. 1, Vol. 12, pp. 1–28). Combinatorial Optimization. Netherland: Springer. doi:10.1007/0-306-48213-4_1

Quinlan, J. R. (1993). *C4.5: programs for machine learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Raschke, R. L., Krishen, A. S., Kachroo, P., & Maheshwari, P. (2012). A combinatorial optimization based sample identification method for group comparisons. *Journal of Business Research*. (to appear). doi:10.1016/j.jbusres.2012.02.024

Reeves, C. R. (1995). A genetic algorithm for flowshop sequencing. *Computers & Operations Research*, *22*(1), 5–13. doi:10.1016/0305-0548(93)E0014-K

Reinartz, T. (2002, April). A unifying view on instance selection. *Data Mining and Knowledge Discovery*, *6*(2), 191–210. doi:10.1023/A:1014047731786

Reinelt, G. (1994). Candidate sets. In *The traveling salesman: computational solutions for TSP applications* (Chap. 5, Vol. 840, pp. 381–412). Lecture Notes in Computer Science. Berlin, Heidelberg: Springer-Verlag. doi:10.1007/3-540-48661-5_5

Rennie, J. D., Shih, L., Teevan, J., & Karger, D. R. (2003). Tackling the poor assumptions of Naive Bayes text classifiers. In *Proceedings of the twentieth international conference on machine learning (icml-2003)* (pp. 616–623). Washington DC, USA: AAAI Press. Retrieved April 4, 2012, from http://aaaipress.org/Papers/ICML/2003/ICML03-081.pdf

Reza Hejazi, S., & Saghafian, S. (2005). Flowshop-scheduling problems with makespan criterion: a review. *International Journal of Production Research*, *43*(14), 2895–2929. doi:10.1080/00207540500056417

Rice, J. R. (1976). The algorithm selection problem. In M. Rubinoff & M. C. Yovits (Eds.), (Vol. 15, pp. 65–118). Advances in Computers. Elsevier. doi:10.1016/S0065-2458(08)60520-3

Richardson, L. F. (1961). The problem of contiguity: an appendix to statistic of deadly quarrels. *General systems: Yearbook of the Society for the Advancement of General Systems Theory*, *6*, 139–187.

Rish, I. (2001, November). *An empirical study of the naive bayes classifier* (tech. rep. No. RC 22230 (W0111-014)). IBM Research Division. Yorktown Heights, NY, USA. Retrieved April 4, 2012, from http://www.research.ibm.com/people/r/rish/papers/RC22230.pdf

Rubinstein, R. Y. (1997). Optimization of computer simulation models with rare events. *European Journal of Operational Research*, *99*(1), 89–112. doi:10.1016/S0377-2217(96)00385-2

Rubinstein, R. Y. (1999). The cross-entropy metod for combinatorial and continuous optimization. *Methodology and Computing in Applied Probability*, *1*(2), 127–190. doi:10.1023/A:1010091220143

Rudolph, G. (1994, January). Convergence analysis of canonical genetic algorithms. *IEEE Transactions on Neural Networks*, *5*(1), 96–101. doi:10.1109/72.265964

Ruiz-Vanoye, J. A., Pérez-Ortega, J., R., R. A. P., Díaz-Parra, O., Frausto-Solís, J., Huacuja, H. J. F., … F., J. A. M. (2011). Survey of polynomial transformations between NP-complete problems. *Journal of Computational and Applied Mathematics*, *235*(16), 4851–4865. doi:10.1016/j.cam.2011.02.018

Sahni, S. (1974, December). Computationally related problems. *SIAM Journal on Computing*, *3*(4), 262–279. doi:10.1137/0203021

Sha, D. Y., & Liu, C.-H. (2005, June). Using data mining for due date assignment in a dynamic job shop environment. *The International Journal of Advanced Manufacturing Technology*, *25*(11), 1164–1174. doi:10.1007/s00170-003-1937-y

Sima, Q. (2010). *Records of the grand historian* (2nd) (Y.-W. Wang, Ed.). The Twenty-four Histories of Baina Edition. Taipei, Taiwan: Commercial Press (Taiwan), Ltd. (Original work published 91B.C.).

Smith, D. E. (1929). *A source book of mathematics* (G. D. Walcott, Ed.). Source Books in ths Histrory of the Sciences. New York, NY, USA: McGraw-Hill Book Company, Inc. Retrieved April 4, 2012, from http://archive.org/details/sourcebookinmath00smit

Smith-Miles, K. A. (2008a, December). Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Computing Surveys*, *41*(1), 6:1–6:25. doi:10.1145/1456650.1456656

Smith-Miles, K. A. (2008b, June). Towards insightful algorithm selection for optimisation using meta-learning concepts. In *2008 IEEE international*

joint conference on neural networks (ijcnn 2008) (pp. 4118–4124). doi:10.1109/IJCNN.2008.4634391

Smith, R. B. (1913). *Carthage and the Carthaginians*. London, UK: Longmans, Green, and Co. Retrieved April 4, 2012, from http://www.archive.org/details/CarthageAndTheCarthaginians

Soare, R. I. (1996, September). Computability and recursion. *The Bulletin of Symbolic Logic*, *2*(3), 284–321. doi:10.2307/420992

Ssu-ma, C. (1994). Sun tzu and Wu Ch'i (T.-f. Cheng, Z. Lu, W. H. Nienhauser & R. Reynolds, Trans.). In W. H. Nienhauser (Ed.), *The memoirs of pre-han china* (Chap. 5, Vol. 7, pp. 39–40). The Grand Scribe's Records. Bloomington, Indiana, USA: Indiana University Press. (Original work published 91B.C.).

Stillwell, J. (2010). *Mathematics and its history* (3rd). Undergraduate Texts in Mathematics. New York, NY, USA: Springer-Verlag. doi:10.1007/978-1-4419-6053-5

Stützle, T. (1998). *Local search algorithms for combinatorial problems — analysis, improvements, and new applications*. (Doctoral dissertation, Department of Computer Science, University of Essex, Colchester, UK).

Stützle, T., & Hoos, H. H. (1997, April). MAX-MIN ant system and local search for the traveling salesman problem. In *Proceedings of the IEEE international conference on evolutionary computation (icec'97)* (pp. 309–314). Indianapolis, IN, USA. doi:10.1109/ICEC.1997.592327

Suykens, J., & Vandewalle, J. (1999). Least squares support vector machine classifiers. *Neural Processing Letters*, *9*, 293–300. doi:10.1023/A:1018628609742

Taillard, E. (1990). Some efficient heuristic methods for the flow shop sequencing problem. *European Journal of Operational Research*, *47*(1), 65–74. doi:10.1016/0377-2217(90)90090-X

Taillard, E. (1993). Benchmarks for basic scheduling problems. *European Journal of Operational Research*, *64*(2), 278–285. doi:10.1016/0377-2217(93)90182-M

Talukdar, S., Baerentzen, L., Gove, A., & De Souza, P. (1998, December). Asynchronous teams: cooperation schemes for autonomous agents. *Journal of Heuristics*, *4*(4), 295–321. doi:10.1023/A:1009669824615

Tay, J. C., & Ho, N. B. (2008). Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems. *Computers & Industrial Engineering*, *54*(3), 453–473. doi:10.1016/j.cie.2007.08.008

Thompson, S. K. (2002). *Sampling* (2nd). Wiley Series in Probability and Statistics. New York, NY, USA: Wiley-Interscience. doi:10.1002/9781118162934

Tsutsui, S., Pelikan, M., & Goldberg, D. E. (2003, August). *Using edge histogram models to solve permutation problems with probabilistic model-building genetic algorithms* (tech. rep. No. 2003022). Illinois Genetic Algorithms Lab, Department of General Engineering, University of Illinois at Urbana-Champaign. Urbana, Illinois, USA. Retrieved April 4, 2012, from http://illigal.org/2003/04/20/using-edge-histogram-models-to-solve-permutation-problems-with-probabilistic-model-building-genetic-algorithms/

Turing, A. M. (1936). On computable numbers, with an application to the Entscheidungsproblem. In *Proceedings london mathematical society* (Vol.

42, *4*, pp. 230–265). 2. Retrieved April 4, 2012, from http://www.cs.virginia.edu/~robins/Turing_Paper_1936.pdf

van der Bruggen, L. J. J., Lenstra, J. K., & Schuur, P. C. (1993, August). Variable-depth search for the single-vehicle pickup and delivery problem with time windows. *Transportation Science*, *27*(3), 298–311. doi:10.1287/trsc.27.3.298

Vázquez-Rodríguez, J. A., & Ochoa, G. (2011). On the automatic discovery of variants of the NEH procedure for flow shop scheduling using genetic programming. *62*(2), 381–396. doi:10.1057/jors.2010.132

Vázquez-Rodríguez, J. A., & Petrovic, S. (2010). A new dispatching rule based genetic algorithm for the multi-objective job shop problem. *Journal of Heuristics*, *16*, 771–793. doi:10.1007/s10732-009-9120-8

Vilalta, R., & Drissi, Y. (2002). A perspective view and survey of meta-learning. *Artificial Intelligence Review*, *18*, 77–95. doi:10.1023/A:1019956318069

Virgil. (1893). *The æneid of Vergil* (J. Rhoades, Trans.). London, UK: Longmans, Green, and Co. (Original work published 29-19B.C.). Retrieved April 4, 2012, from http://scans.library.utoronto.ca/pdf/1/5/aeneidbooks16tra00virguoft/aeneidbooks16tra00virguoft.pdf

Virgil. (1904). *Virgil's Aeneid* (C. E. Bennett, Ed.). Bennett's Latin series. Boston, MA: Allyn and Bacon. (Original work published 29-19B.C.). Retrieved April 4, 2012, from http://archive.org/details/aeneid00virggoog

Wikipedia. (2012a). How long is the coast of britain? statistical self-similarity and fractional dimension — wikipedia, the free encyclopedia. (page version ID: 482513336). Retrieved April 4, 2012, from http://en.wikipedia.org/w/index.php?title=How_Long_Is_the_Coast_of_Britain%3F_Statistical_Self-Similarity_and_Fractional_Dimension&oldid=482513336

Wikipedia. (2012b). NP (complexity) — wikipedia, the free encyclopedia. (page version ID: 484061972). Retrieved April 4, 2012, from http://en.wikipedia.org/w/index.php?title=NP_(complexity)&oldid=484061972

Williamson, D. P., Hall, L. A., Hoogeveen, J. A., Hurkens, C. A. J., Lenstra, J. K., Sevast'janov, S. V., & Shmoys, D. B. (1997, April). Short shop schedules. *Operations Research*, *45*(2), 288–294. doi:10.1287/opre.45.2.288

Witten, I. H., & Frank, E. (2005). *Data mining: practical machine learning tools and techniques* (2nd). Morgan Kaufmann Series in Data Management Systems. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Woeginger, G. J. (2003). Combinatorial optimization — eureka, you shrink! In M. Jünger, G. Reinelt & G. Rinaldi (Eds.), (Chap. Exact algorithms for NP-hard problems: a survey, Vol. 2570, pp. 185–207). Lecture Notes in Computer Science. New York, NY, USA: Springer-Verlag New York, Inc. doi:10.1007/3-540-36478-1_17

Wolpert, D. H., & Macready, W. G. (1997, April). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, *1*(1), 67–82. doi:10.1109/4235.585893

Woodhouse, R. (1964). *A treatise on isoperimetrical problems and the calculus of variations* [A history of the calculus of variations in the eighteenth century]. Bronx, NY, USA: AMS Chelsea Publishing. (Original work published 1810).

Xue, F., Chan, C. Y., Ip, W. H., & Cheung, C. F. (2011). A learning-based variable assignment weighting scheme for heuristic and exact searching in Euclidean traveling salesman problems. *NETNOMICS: Economic*

Research and Electronic Networking, *12*, 183–207. doi:10.1007/s11066-011-9064-7

Xue, F., & Fan, W. (2007). Multi-agent optimization design for multi-resource job shop scheduling problems. In S. Huang, L. Heutte & M. Loog (Eds.), *Advanced intelligent computing theories and applications: with aspects of artificial intelligence* (Vol. 4682, pp. 1193–1204). Lecture Notes in Computer Science. Germany: Springer-Verlag. doi:10.1007/978-3-540-74205-0_123

Xu, L., Hutter, F., Hoos, H. H., & Leyton-Brown, K. (2008, June). SATzilla: portfolio-based algorithm selection for SAT. *Journal of Artificial Intelligence Research*, *32*(1), 565–606. Retrieved April 4, 2012, from http://www.aaai.org/Papers/JAIR/Vol32/JAIR-3214.pdf

Xu, L., Krzyżak, A., & Suen, C. Y. (1992, June). Methods of combining multiple classifiers and their applications to handwriting recognition. *IEEE Transactions on Systems, Man and Cybernetics*, *22*(3), 418–435. doi:10.1109/21.155943

Yousef, M., Nebozhyn, M., Shatkay, H., Kanterakis, S., Showe, L. C., & Showe, M. K. (2006). Combining multi-species genomic data for microRNA identification using a naïve bayes classifier. *Bioinformatics*, *22*(11), 1325–1334. doi:10.1093/bioinformatics/btl094

Zaki, M. J., Parthasarathy, S., Li, W., & Ogihara, M. (1997). Evaluation of sampling for data mining of association rules. In *Proceedings of the 7th international workshop on research issues in data engineering (ride '97)* (pp. 42–50). Washington, DC, USA: IEEE Computer Society. doi:10.1109/RIDE.1997.583696

Zhang, G. P. (2000, November). Neural networks for classification: a survey. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, *30*(4), 451–462. doi:10.1109/5326.897072

Zhang, W., & Looks, M. (2005). A novel local search algorithm for the traveling salesman problem that exploits backbones. In *Proceedings of the 19th international joint conference on artificial intelligence* (pp. 343–348). IJCAI'05. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. Retrieved April 4, 2012, from http://www.cs.wustl.edu/~zhang/publications/bgtsp.pdf

Zobolas, G. I., Tarantilis, C. D., & Ioannou, G. (2009). Minimizing makespan in permutation flow shop scheduling problems using a hybrid metaheuristic algorithm. *Computers & Operations Research*, *36*(4), 1249–1267. doi:10.1016/j.cor.2008.01.007