Summer 2012

Incorporating Memory and Learning Mechanisms Into Meta-RaPS

Arif Arin
Old Dominion University

INCORPORATING MEMORY AND LEARNING MECHANISMS

INTO META-RAPS

by

Arif Arin

B.S. August 1995, Air Force Academy, Turkey
M.S. March 2002, Air Force Institute of Technology

A Dissertation Submitted to the Faculty of
Old Dominion University in Partial Fulfillment of the
Requirements for the Degree of

DOCTOR OF PHILOSOPHY

ENGINEERING MANAGEMENT

OLD DOMINION UNIVERSITY
August 2012

Approved by:

Ghaith Rabadi (Director)

Resit Unal (Member)

Holly Handley (Member)

Okay Isik (Member)

ABSTRACT

INCORPORATING MEMORY AND LEARNING MECHANISMS
INTO META-RAPS

Arif Arin
Old Dominion University, 2012
Director: Dr. Ghaith Rabadi

Due to the rapid increase of dimensions and complexity of real life problems, it has become more difficult to find optimal solutions using only exact mathematical methods. The need to find near-optimal solutions in an acceptable amount of time is a challeng when developping more sophisticated approaches. A proper answer to this challenge can be through the implementation of metaheuristic approaches. However, a more powerful answer might be reached by incorporating intelligence into metaheuristics.

Meta-RaPS (Metaheuristic for Randomized Priority Search) is a metaheuristic that creates high quality solutions for discrete optimization problems. It is proposed that incorporating memory and learning mechanisms into Meta-RaPS, which is currently classified as a memoryless metaheuristic, can help the algorithm produce higher quality results.

The proposed Meta-RaPS versions were created by taking different perspectives of learning. The first approach taken is Estimation of Distribution Algorithms (EDA), a stochastic learning technique that creates a probability distribution for each decision variable to generate new solutions. The second Meta-RaPS version was developed by utilizing a machine learning algorithm, Q Learning, which has beeen successfully applied to optimization problems whose output is a sequence of actions. In the third Meta-RaPS

version, Path Relinking (PR) was implemented as a post-optimization method in which the new algorithm learns the "good" attributes by memorizing best solutions, and follows them to reach better solutions. The fourth proposed version of Meta-RaPS presented another form of learning with its ability to adaptively tune parameters. The efficiency of these approaches motivated us to redesign Meta-RaPS by removing the improvement phase and adding a more sophisticated Path Relinking method. The new Meta-RaPS could solve even the largest problems in much less time while keeping up the quality of its solutions.

To evaluate their performance, all introduced versions were tested using the 0-1 Multidimensional Knapsack Problem (MKP). After comparing the proposed algorithms, Meta-RaPS PR and Meta-RaPS Q Learning appeared to be the algorithms with the best and worst performance, respectively. On the other hand, they could all show superior performance than other approaches to the 0-1 MKP in the literature.

This dissertation is dedicated to my family who have always been with me during this long journey.

ACKNOWLEDGMENTS

Although only my name is written on the cover of this dissertation, indeed, I am only one of the many people who have contributed to this work. Foremost, I would like to express my sincere gratitude to my advisor, Dr. Ghaith Rabadi, for his continuous support, patience, motivation, enthusiasm, and knowledge. He gave me the freedom to explore on my own, and, at the same time, the guidance to recover when my steps faltered. His guidance helped me during the time of the research and writing of this dissertation. I have been amazingly lucky to be his student.

I would like to thank my doctoral committee members for their valuable input and discussions; Professor Resit Unal, Chair of Department of Engineering Management and Systems Engineering, Dr. Murat Ermis and Dr. Okay Isik from the Turkish Air Force Academy, for insightful comments and scholarly inputs. I owe sincere thankfulness to Dr. Reinaldo J. Moraga for his interesting ideas and inspiring me to enter into this exciting research field with Dr. Ghaith Rabadi.

I am also thankful to the system server staff who maintained all the machines so efficiently that I could obtain tha data to complete many parts of this work.

I would like to thank Professor Oktay Baysal, Dean, and Professor A.Osman Akan, Associate Dean of the Frank Batten College of Engineering and Technology, for their support and their valuable contributions on creating the cooperation between Old Dominion University and Aeronautics and Space Technologies Institute. I deeply appreciate the financial support from Turkish Air Force.

I would like to show my gratitude to my friends and colleagues I had the privilege to enjoy their friendship. I am so grateful for their precious welcome and hospitality I have received.

Finally, I am truly indebted and thankful to my wife, Belma, for her consistent support, encouragement, tolerance and love. Her immovable faith in me enabled me to have strength necessary to complete this work. I owe a lot to my other family members, who encouraged and helped me at every stage of my personal and academic life, and longed to see this achievement come true.

NOMENCLATURE

ACO	Ant Colony Optimization
AI	Artificial Intelligence
ALS	Adaptive Learning Search
AMP	Adaptive Memory Programming
AP	Adaptive Parameter
BCS	Best Constructed Solution
BBO	Black Box Optimization
CL	Candidate List
CMean	Mean of Constructed Solutions
COMSOAL	Computer Method of Sequencing Operations for Assembly Lines
CS	Constructed Solution
DGR	Dynamic Greedy Rule
DOE	Design of Experiments
DP	Dynamic Programming
EA	Evolutionary Algorithms
EDA	Estimation of Distribution Algorithms
EP	Evolutionary Programming
ES	Evolution Strategies
GA	Genetic Algorithm
GP	Genetic Programming
GRASP	Greedy Randomized Adaptive Search Procedure
i%	Improvement Percentage
I	Number of Iteration

IMean	Mean of Best Improved Solutions
IS	Improved Solution
Meta-RaPS	Meta-heuristic for Randomized Priority Search
MKP	Multidimensional Knapsack Problem
p%	Priority Percentage
PR	Path Relinking
PSO	Particle Swarm Optimization
r%	Restriction Percentage
RL	Reinforcement Learning
SA	Simulated Annealing
RS	Reactive Search
SS	Scatter Search
TS	Tabu Search
WCS	Worst Constructed Solution

TABLE OF CONTENTS

Page

LIST OF TABLES

Table Page

Table Page

Table Page

LIST OF FIGURES

Figure Page

CHAPTER 1

INTRODUCTION

"If we are to achieve results never before accomplished,
we must expect to employ methods never before attempted."
Sir Francis Bacon

In our constantly changing environment, we always adapt ourselves to different situations that we encounter in life. Instead of "hardwiring" (Alpaydın, 2004) all types of behavior into us, we learn the best strategies in certain cases and store them in our brain to call when similar situations arise again.

Learning, according to Fogel (1995), is an intelligent process in which the basic unit of mutability is the idea. "Good" adaptive ideas are maintained, much as good genes increase in a population, while poor ideas are forgotten. In insect societies, this only requires the evaporation of pheromone trails; in humans it requires time for actual forgetting (Kennedy, et al., 2001). In a similar manner, memory and learning mechanisms in metaheuristics can learn and remember "good" ideas related to the search process to make it possible to create high quality solutions for optimization problems by utilizing this information. In the problem solving arena, the definition of intelligence emerges in metaheuristics via memory and learning. Many successful metaheuristics employ "intelligent" procedures to obtain high quality solutions for optimization problems.

1.1 Metaheuristics

With the growing complexity of today's large scale problems, it has become more difficult to find optimal solutions using only exact mathematical methods. Due to computational efficiency concerns, the need to find near-optimal solutions in an acceptable amount of time requires using heuristic approaches. Birattari (2009) defines a heuristic as "a generic algorithmic template that can be used for finding high quality solutions of hard combinatorial optimization problems" (page VII). Heuristic approaches have already proved themselves in many large scale optimization problems by offering near-optimal solutions where it is difficult to find optimal solutions by other approaches.

In theory, there is a chance to find the optimum solution by implementing heuristic methods. However, often being trapped in local optima can move the heuristics away from the optimum solution. Metaheuristics or "modern heuristics" confront this challenge by adding strategies and mechanisms to avoid local optima to the construction and local search mechanisms already existing in heuristics (Moraga, 2009). Glover & Laguna (1997) define metaheuristics as "a master strategy that guides and modifies other heuristics to produce solutions beyond those that are normally generated in a quest for local optimality" (page 17).

Glover & Laguna (1997) introduced a classification method for metaheuristics depending on three design choices: the use of adaptive memory, the type of neighborhood exploration used, and the number of current solutions carried from one iteration to the next. The metaheuristic classification notation can be illustrated in the form a|b|c. If the metaheuristic has adaptive memory, the first letter, a, will be *A*, and *M* if the method is memoryless. Depending on the neighborhood mechanism, the second letter, b, will be *N*

for somehow systematic neighborhood search, and *S* for using random sampling. The third letter, c, can be *1* for a single-solution approach or *P* for a population-based approach with population size of P. The classification method for metaheuristics is summarized in Table 1.

Table 1. Classification Method for Metaheuristics

a		b		c	
Use of Adaptive Memory		Type of Neighborhood		Number of Solutions Carried at Iterations	
A	M	N	S	1	P
Adaptive Memory	Memoryless	Systematic Neighborhood Search	Random Sampling	Single Solution	Population Size of P

1.2 Meta-RaPS as a Memoryless Metaheuristic

Besides exact mathematical methods, metaheuristics methods are quite promising approaches in solving optimization problems especially in terms of their results, the size of the problem dealt with and computational effort consumed. Although, as a metaheuristic, Meta-heuristic for Randomized Priority Search (Meta-RaPS) has been generating very promising solutions when applied to optimization problems, Meta-RaPS is currently classified as a memoryless metaheuristic. The reason for this classification is that there is no memory mechanism in Meta-RaPS to memorize the information created in the solution process, nor a learning mechanism to learn the structure of this process in making future decisions.

In many cases it has been observed that, memory and learning mechanisms increase the effectiveness of the solution process, and as a result, the solution quality. Therefore, by incorporating some memory and learning tools into Meta-RaPS, the algorithm is expected to use this information in creating higher quality solutions. To reach this goal, four algorithms from different fields are selected: Estimation of Distribution Algorithms (EDA) as a stochastic approach and Q learning as a machine learning approach are the first two algorithms to offer their memory and learning abilities to Meta-RaPS. The third algorithm is Path Relinking (PR), a post-optimization method that learns the "good" attributes of the best solutions, and follows them to reach better solutions. The fourth algorithm is selected from Adaptive Parameter (AP) tuning area that plays a key role in a metaheuristic's performance. The ultimate goal of this research is to redesign Meta-RaPS to become more "intelligent" with the ability to memorize and learn in order to reach higher quality solutions more efficiently by removing the improvement phase in Meta-RaPS if possible.

The proposed algorithms share a common characteristic of employing some memory and learning mechanisms in their search spaces. To evaluate the performance of the proposed algorithms, the 0-1 Multidimensional Knapsack Problem (MKP), which is a special case of the general linear 0-1 integer programming problem with nonnegative coefficients, will be used as testbed.

CHAPTER 2

META-RAPS AND 0-1 MULTIDIMENSIONAL KNAPSACK PROBLEM

The two main actors having constant roles in each process of incorporating memory and learning mechanisms are Meta-RaPS (Metaheuristic for Randomized Priority Search) and the 0-1 Multidimensional Knapsack Problem (MKP). As a memoryless metaheuristic, various intelligent versions of Meta-RaPS will be proposed and the 0-1 MKP will be used as their testbad in the applications that will be presented in this research.

2.1 Meta-RaPS, a Simple and Powerful Metaheuristic

Meta-RaPS is one of the randomized search metaheuristics, and stands for "Meta-heuristic for Randomized Priority Search". Moraga, et al. (2006) defines Meta-RaPS as "generic, high level search procedures that introduce randomness to a construction heuristic as a device to avoid getting trapped at a local optimal solution" Page (10-8). Meta-RaPS combines the mechanisms of priority rules, randomness, and sampling.

Meta-RaPS is currently classified as a memoryless metaheuristic and can benefit from existing memory and learning mechanisms to increase its effectiveness.

2.1.2 Literature Review

Meta-RaPS produced high quality solutions when applied to discrete optimization problems, such as the Resource Constrained Project Scheduling Problem (DePuy & Whitehouse, 2001) and the Vehicle Routing Problem (Moraga, 2002).

DePuy, et al. (2005) aimed to develop a simple method to find good solutions to traveling salesman problems (TSP). The Meta-RaPS approach was introduced as a method of incorporating randomness in established TSP priority schemes. The Meta-RaPS approach outperformed most other solution methodologies in terms of percent difference from optimal. Additionally, an industry case study that incorporates Meta-RaPS TSP in a large truck route assignment model is presented. The company estimates a more than 50% reduction in engineering time and over $2.5 million annual savings in transportation costs using the automated Meta-RaPS TSP tool compared to their current method.

Moraga, et al. (2005) presented Meta-RaPS approach for the 0-1 Multidimensional Knapsack Problem (0-1 MKP). The Meta-RaPS incorporated with a greedy algorithm called the Dynamic Greedy Rule (DGR) outperformed many other solution methodologies, such as simulated annealing, tabu search, genetic algorithms, and 0-1 MKP heuristics, in terms of differences from the optimal value and number of optimal solutions obtained. They also noted that the performance of Meta-RaPS DGR was not quite as good as that of Chu and Beasley's (1998) genetic algorithm or Bertsimas and Demir's (2002) approximate dynamic programming for the largest problem sizes, and further investigation can be done to improve the solution quality for these large problems.

Rabadi, et al. (2006) introduced Meta-RaPS for the unrelated parallel machine scheduling problem (PMSP) with machine-dependent and sequence-dependent setup times to minimize the makespan. According to the results, Meta-RaPS found all optimal

solutions for small problems and for larger problems it outperformed the solutions obtained by the existing heuristic the Partitioning Heuristic by Al-Salem (2004).

The Set Covering Problem (SCP) was another optimization problem selected as the application for Meta-RaPS where it is found that these randomization methods work well (Lan & DePuy, 2006; Lan, DePuy & Whitehouse, 2007). Lan, DePuy & Whitehouse (2007) developed an effective heuristic to solve the set covering problem (SCP) by applying Meta-RaPS. In addition to basic principles of Meta-RaPS, they penalized the worst columns if the solution searching space is highly condensed to enhance the performance of the basic Meta-RaPS. They reported this algorithm was the best in solution quality among all the heuristic algorithms available in the literature for solving the test instances in the OR Library.

Hepdogan, et al. (2009) applied Meta-RaPS algorithm to the early/tardy single machine scheduling problem with common due date and sequence-dependent setup times (ETP). In this case, the Smallest Adjusted Processing Time (SAPT) rule is modified by Meta-RaPS with its ability of randomness. When comparing Meta-RaPS ETP with a simulated annealing (SA) and a hybrid approach Smallest Adjusted Processing Time – SA (SAPT-SA) technique, they observed that Meta-RaPS produced better solutions in terms of percent difference from optimal and in computation time.

Kaplan, et al. (2010) used Meta-RaPS approach to solve the Aerial Refueling Scheduling Problem (ARSP), a real world problem that requires high quality solutions in an acceptable time frame. ARSP can be defined as determining the refueling completion times for each fighter aircraft wing (job) on multiple tankers (machines) and therefore can be modeled as a parallel machine scheduling with release times and due dates to

minimize the total weighted tardiness. In their study, Meta-RaPS showed to be a promising metaheuristic with its simplicity and effectiveness to find high quality solutions for the ARSP. Kaplan and Rabadi (forthcoming) also presented a Simulated Annealing and Meta-RaPS algorithm for the ARSP with due date-to-deadline windows and release time.

Garcia and Rabadi (2011) developed a new algorithm based on Meta-RaPS for solving the parallel multiple-area spatial scheduling problem with release times. Meta-RaPS presented better performance on a set of highly diverse benchmark problems when compared to the results obtained by a MIP model solved with CPLEX; it was very effective and in most cases tied or outperformed CPLEX requiring less than 20% of the computational time used by CPLEX.

Different from other authors, Hepdogan, et al. (2008) investigated the problem of setting parameters using Meta-RaPS. They presented two different dynamic parameter setting methods, Nonparametric Genetic Algorithms (NPGA) and Reactive Search (RS), for Meta-RaPS while a solution is being found. These parameter setting methods were used to set the parameters of Meta-RaPS to solve 0-1 MKP and ETP.

2.1.3 Meta-RaPS - Related Algorithms

Meta-RaPS is based on the "Computer Method of Sequencing Operations for Assembly Lines" (COMSOAL). Meta-RaPS, however, is a general form of GRASP (greedy randomized adaptive search procedure) which is a greedy metaheuristic to solve combinatorial optimization problems. In the following sections both algorithms are discussed due to their relevance to the Meta-RaPS.

2.1.3.1 COMSOAL

Meta-RaPS is based on the COMSOAL, which is an iterative computer heuristic introduced by Arcus (1966) for balancing large complex machine-paced assembly lines. COMSOAL originated in an industrial operations research project and was in use at some factories of the Chrysler Corporation in an early form (Arcus, 1966).

COMSOAL generates a list of activities (candidate list) to be scheduled next. In order to be selected for the candidate list, an activity must have all its predecessor activities completed and there must be enough resources available to perform the activity. The next activity is selected randomly to be scheduled from this candidate list. This iterative process continues until all activities have been scheduled and a feasible schedule is obtained. After several iterations of this procedure the best solution found is reported (DePuy & Whitehouse, 2000).

DePuy and Whitehouse (2000) discussed the adaptation of the COMSOAL approach to the resource allocation problem as well as a designed experiment used to investigate the appropriateness of COMSOAL for a known set of resource allocation test problems. DePuy and Whitehouse (2001) modified COMSOAL to the resource constrained project scheduling problem (RCPSP). The Modified COMSOAL was using priority schemes intermittently with a random selection technique, and outperformed the other heuristics in terms of the average and maximum percentage difference from optimal.

According to DePuy, et al. (2001), although the modified versions of COMSOAL keep the fundamental ideas of Arcus (1966), in practice, the created versions of

COMSOAL differ considerably from the original one, thus leading to the development of Meta-RaPS.

2.1.3.2 GRASP

GRASP is an iterative greedy heuristic introduced by Feo and Resende (1989) to solve combinatorial optimization problems. The GRASP algorithm consists of two phases: construction and local search. The feasible solutions constructed in the first phase are not guaranteed to be locally optimal. Usually a local search is performed to attempt to improve each constructed solution in the second phase.

In the construction phase of GRASP, the next components or activities are selected according to their greedy evaluation function which calculates their incremental cost if they are incorporated with the current components in the partial solution. The best components or activities, i.e., the ones with the minimum incremental costs, are collected to create the restricted candidate list (RCL) from which the components or activities are chosen randomly to incorporate. While the first phenomenon indicates greedy attributes, the latter shows probabilistic attribute of the algorithm. After incorporating the components or activities in the partial solution, the RCL is updated by calculating the new incremental costs of the components or activities left in the process which points out the adaptive attribute of the GRASP (Resende & Ribeiro, 2003). Because the solution generated in the construction phase is not usually optimal, the local search phase becomes part of the solution process to improve the solutions. Besides local searches, any single-solution heuristic can also be employed as an improvement means for GRASP (Talbi, 2009).

The main parameters for GRASP are related to the stopping criterion and the quality of the solutions in the RCL. Increasing the number of the iterations will also increase the probability of reaching better solutions but consuming more computer time; on the other hand, the second parameter will help improve the quality of the best solutions in the RCL. One of the approaches to reach the latter goal is value-based criteria by using a threshold parameter $\alpha \in [0, 1]$ which controls the greediness and the randomness in the search process. If c(a) is the incremental cost when the component or activity *a* is incorporated in the current partial solution; and c^{min} and c^{max} are denoted as the smallest and the largest incremental costs, respectively, the RCL can be formed from the component or activity *a* associated with c(a) selected as in (2.1):

$$c(a) \in [c^{min}, c^{min} + \alpha(c^{max} - c^{min})]. \quad (2.1)$$

The parameter α for GRASP specifies the balance between intensification using more greediness attribute and diversification using more randomness attribute. For the case of $\alpha = 0$ GRASP runs as a greedy algorithm, and for the case $\alpha = 1$ GRASP runs as a random algorithm.

Although GRASP generates solutions by introducing randomness to a greedy heuristic like Meta-RaPS, it does not implement any probabilistic priority to the best solutions (Hepdogan, et al., 2009). The iterations in GRASP algorithm are totally independent, and there is no search memory. GRASP is classified as M|S|1.

2.1.4 Meta-RaPS Algorithm

Like GRASP, Meta-RaPS is a two-phase metaheuristic: a constructive phase to create feasible solutions and an improvement phase to improve them. In the constructive

phase, a solution is built by repeatedly adding feasible components or activities to the current solution in an order that is based on their priority rules until the stopping criterion is satisfied. Generally, solutions obtained by implementing only constructive algorithms can reach mostly local optima, which can be avoided in Meta-RaPS by employing randomness in the constructive phase.

Meta-RaPS uses four parameters: number of iterations (I), the priority percentage (p%), the restriction percentage (r%), and the improvement percentage (i%). Meta-RaPS does not select the component or activity with the best priority value in every iteration, nor does it select the one with the lowest incremental cost. Instead, the algorithm may randomly accept an activity or component with a good priority value, but not necessarily the best one. The parameter p% is used to decide the percentage of time a component or activity with the best priority value will be added to the current partial solution, and 100% – p% of the time it will be randomly selected from a candidate list (CL) containing "good" components or activities. The CL is created by including items whose priority values are within r% of the best priority value. The CL is therefore created using equations (2.2) and (2.3) where P_b is the component or activity with the best priority value and F is the set of feasible components or activities (Lan, et al., 2007):

$$CL = \{ i : i \in F \text{ and } P_i \leq P_b \cdot (1 + r\%) \} \text{ for minimization.} \quad (2.2)$$

$$CL = \{ i : i \in F \text{ and } P_i \geq P_b \cdot (1 - r\%) \} \text{ for maximization.} \quad (2.3)$$

In the construction phase, the level of the randomness is adjusted by controlling the values of the parameters p% and r% where smaller values of p% and larger values of r% will randomize the search more. The construction phase of Meta-RaPS is completed when a feasible solution is produced.

The improvement phase is performed if the feasible solutions generated in the construction phase are within i% of the best unimproved solution value from the preceding iterations. For the feasible solution to be improved in this phase, it must be determined whether its objective function value Z satisfies the requirements in (2.4) and (2.5) where Z^* is the solution with the best objective function value obtained in the construction phase (Lan, et al., 2007). Meta-RaPS pseudocode is shown in Figure 1.

$$Z \leq Z^* \cdot (1 + i\%) \text{ for minimization.} \quad (2.4)$$

$$Z \leq Z^* \cdot (1 - i\%) \text{ for maximization.} \quad (2.5)$$

The quality of the solution created by Meta-RaPS is heavily dependent to its parameters, especially the number of iterations and the improvement percentage. However, increasing the values of these parameters will also increase the need for more computational time (Hepdogan, et al., 2009).

Meta-RaPS can be compared with COMSOAL and GRASP depending on the values of the parameters in Meta-RaPS. While using 0 for the priority percentage, an infinitely large restriction percentage, and 0 for the improvement percentage will imitate COMSOAL; 0 for the priority percentage and 100 for the improvement percentage will simulate GRASP (Moraga, et al., 2005). Because, in Meta-RaPS these three parameters can take different values other than 0 and 100%, it exhibits much more flexibility over COMSOAL and GRASP.

Meta-RaPS procedure is simple and effective procedure with only two main parameters to be set. The simple nature of Meta-RaPS coupled with its ability to generate high quality solutions, makes Meta-RaPS a good metaheuristic method for combinatorial optimization problems (Hepdogan, et al., 2009).

```
For iteration ≤ I
While (feasible solution is not constructed)
        Find priority value for each feasible activity
        Find best priority value
        If rnd() ≤ priority% then
                add item with best priority value to solution
        Else create CandidateList from feasible activities with
                priority values ≥ Limit
                Limit = MinimumPriority +
                        restriction% · (MaximumPriority - MinimumPriority)
                Choose randomly an item from CandidateList and add to solution
End While
Δ = BestConstructedSolution · improvement%
If ConstructedSolution ≥ Δ then improve
If ImprovedSolution > BestImprovedSolution then
   Assign ImprovedSolution as BestImprovedSolution
End For
Report BestImprovedSolution
```

Figure 1. Meta-RaPS Pseudo Code

DePuy, et al. (2001) expressed the advantages of the Meta-RaPS over other metaheuristics. According to them;

- Run times for Meta-RaPS are not significantly affected by the size of the problem,
- Meta-RaPS is easy to understand and to implement (i.e. write computer code),

- Meta-RaPS generates a feasible solution at every iteration.

Many real world optimization problems require a deep understanding of mathematical and computer programming. In this aspect, since it is easy to understand and to put into application, and it can create good results in a reasonable amount of time, Meta-RaPS should be particularly attractive to industrial practitioners too (Moraga, et al., 2005).

2.2 0-1 Multidimensional Knapsack Problem

The multidimensional 0–1 knapsack problem (0-1 MKP) is a special case of general linear 0–1 programs. The MKP can be considered as a subproblem of other optimization problems as the multidemand multidimensional knapsack problem (Wilbaut & Hanafi, 2008). In the literature there are different names used for the MKP: m-dimensional knapsack, problem, multidimensional knapsack problem, multiknapsack problem, multiconstraint 0–1 knapsack problem (Freville, 2004). The name *multidimensional 0-1 knapsack problem* that will be used here was mentioned first by Weingartner and Ness (1967). The MKP is often used as a platform to evaluate new metaheuristics and will therefore be used in this research to evaluate the effectiveness of the proposed methods.

2.2.1 Literature Review

The first applications of 0-1 MKP had been presented by Lorie and Savage (1955) and by Manne and Markowitz (1957) as a capital budgeting model. To solve the 0-1 MKP, both exact and approximation algorithms have been used. Exact algorithm includes

enumeration method, graph theoretic approach and dynamic programming (Martello, et al., 2000). The development of exact algorithms began at the same time for both the KP and MKP and included dynamic programming, branch-and-bound network approach, hybridization of dynamic programming and branch-and-bound, special enumeration technique and reduction schemes (Fréville, 2004).

Exact methods for 0-1 MKP are based on dynamic programming (Gilmore & Gomory, 1966; Martello, Pisinger & Toth, 1999; Pisinger, 1997; Weingartner & Ness, 1967), and in branch-and-bound techniques (Gavish & Pirkul, 1985; Martello and Toth, 1988; Pisinger, 1995; Sarin, Karwan & Rardin, 1988; Shih, 1979). Balev, et al. (2008) presented a new dynamic programming based approach to the 0–1 MKP where they used sparse data representation, which decreases memory and time requirements. Meier, Christofides and Salkin (2001) proposed a realistic approach that uses this problem as a subproblem coupled with generalized upper bound constraints. Boussier, et al. (2010) proposed an exact method based on a multi-level search strategy for solving the 0-1 MKP which combines Resolution Search, a Branch and Bound, and a Depth First Search algorithm that exploit efficiently both the reduced costs and the fixed number of item constraints.

Even when recent advances of methods such as branch-and-cut have made the solution of middle size MKP instances possible, increasing the number of constraints makes approximation algorithms necessary. Fleszar and Hindi (2009) presented the heuristics appropriately chosen deterministic or randomly generated constraints imposed on the linear relaxation can be used to partition the solution space effectively, leading to good solutions for 0-1 MKP. The method of Boyer, Elkihel and El Baz (2009) solved the

0-1 MKP with two heuristics. The first heuristic using surrogate relaxation was solved via a modified dynamic-programming algorithm to provide a feasible solution, and a second heuristic was used to improve the bound obtained by exploring some nodes rejected by the modified dynamic-programming algorithm.

James and Nakagama (2005) decomposed the 0-1 MKP into two parts by applying enumeration methods to decrease memory requirement to solve large instances. Wilbaut and Hanafi (2009) presented several convergent algorithms to solve a series of small sub-problems generated by exploiting information obtained from a series of relaxations. Hill, et al. (2012) introduced new problem-size reduction heuristics for the 0-1 MKP. Their heuristics are based on solving a relaxed version of the problem, using the dual variables to formulate a Lagrangian relaxation of the original problem, and then solving an estimated core problem to achieve a heuristic solution to the original problem.

Tabu search approaches proposed by Glover and Kochenberger (1996) and Hanafi and Fréville (1998) alternated between constructive and destructive phases and allowed the visit of infeasible solutions during the search. The tabu search approach of Hanafi and Fréville (1998) was based on strategic oscillation and surrogate constraint information that provides a balance between intensification and diversification strategies. The algorithm of Vasquez and Vimont (2005) combines Linear Programming with an efficient tabu search. He, et al. (2006) proposed a Tabu Search method based on a Double Tabu-List inspired by the conclusion of the cognitive psychology about the human memory system.

The genetic algorithms of Chu and Beasley (1998) and Haul and Vo (1998) obtained good lower bounds for this problem. Haul and Vo (1998) introduced surrogate

relaxations into their genetic algorithm to enhance the process. Khuri, et al. (1994) used a genetic algorithm for solving 0-1 MKP, while Cotta and Troya (1998) combined a constructive heuristic for initialization and a local search method with a genetic algorithm. Kato and Sakawa (2003) introduced the genetic algorithm with decomposition procedures as an approximate solution method for large scale 0-1 MKP utilizing with block angular structures. Cleary and O'Neill (2005) employed grammatical evolution (GE) using different representation schemes.

Moraga, et al. (2005) have applied Meta-RaPS on the 0-1 MKP. The quality of their solutions was close to the one obtained by Chu and Beasley (1998) while using a reasonable computational effort. Hembecker, et al. (2007) applied particle swarm optimization (PSO) for solving 0-1 MKP. Gong, Zhou and Luo (2011) proposed a hybrid artificially glowworm swarm optimization algorithm that utilizes two important strategies, how to select the item based on its unit volume value and the binary glowworm swarm optimization algorithm. In addition to the ant-based approach of Kong (2007), Chiang, et al. (2011) also proposed a novel ant-inspired constructive algorithm, AST-MKP, which adopted a constructive graph for leading artificial ants in making decisions to select effective solution components. Gallardo, et al. (2007) introduced a hybrid model that combines branch-and-bound and memetic algorithms for the 0-1 MKP. Wilbaut, et al. (2009) proposed new iterative heuristics with variable fixation to reduce 0-1 MKP until it becomes sufficiently small to be solved with an exact method in a reasonable CPU time. Other than these approaches, several metaheuristics have been developed including Differential Evolution (Sima & Gülsen, 2005), Simulated Annealing

(Drexl, 1988) and Immune Inspired Algorithm (Maoguo, et al., 2007) focusing on the 0-1 MKP.

The 0-1 MKP has wide range of real-world application areas, such as capital budgeting, allocating processors and databases in distributed computer systems (Gavish & Pirkul, 1985). Cutting stock (Gilmore & Gomory, 1966) and loading problems (Bellman, 1957; Shih, 1979) are also known applications. The MKP has also recently been used to model the daily management of a satellite like SPOT (Vasquez & Hao, 2001), the resource allocation in distributed data processing (Gavish & Pirkul, 1982) and the planning of data-processing programs (Thesen, 1973). The MKP has also been used as a subproblem for solving a multicommodity network optimization problem (Gabrel, Knippel & Minoux, 1999).

Most of the best-known solutions for the instances in the OR Library (Beasley 1990) were obtained by Vasquez and Hao (2001) and Vasquez and Vimont (2005). Vimont, Boussier and Vasquez (2008) obtained several new optimal solutions on hard instances of the OR Library with their implicit enumeration algorithm based on a reduced costs analysis which tends to fix non-basic variables to their exact values.

Pisinger (1995) investigated knapsack problems in general and their categorizations. Chu and Beasley (1998) classified the 0-1 MKP approaches into exact algorithms and heuristic algorithms. Wilbaut and Hanafi (2008) presented a family of knapsack problems with their applications and reviewed appropriate techniques successful in solving these problems. An extensive survey on MKPs can be found in Fréville (2004) and Fréville and Hanafi (2005). The books by Kellerer, Pferschy and Pisinger (2004) and Martello and Toth (1990) provide interesting reviews and useful

references. Wilbaut, et al. (2008) produced a survey paper with an emphasis to effective heuristics and their applications.

2.2.2 Definition of the 0-1 Multidimensional Knapsack Problem

The 0-1 MKP is the generalized form of the classical knapsack problem (KP). In KP there is a knapsack with an upper weight limit b, a set of n items with different profits c_j and weights a_j per item j. The problem is to select the items from the set such that the total profit of the selected items is maximized without exceeding the upper weight limit of the knapsack. If m knapsacks exist, the problem becomes the MKP in which each knapsack has a different upper weight limit b_i, and an item j has a different weight a_{ij} for each knapsack i. The objective is to find a set of items with maximal profit such that the capacity of each knapsack is not exceeded (Gallardo, et al., 2009). The 0-1 MKP can be formulated as in the equations (2.6 - 2.8):

$$\text{Maximize} \quad \sum_{j=1}^{n} c_j x_j . \tag{2.6}$$

$$\text{Subject to} \quad \sum_{j=1}^{n} a_{ij} x_j \le b_i , \; i = 1, \ldots, m; j = 1, \ldots, n. \tag{2.7}$$

$$x_j \in \{0,1\}, \quad j = 1, \ldots, n \tag{2.8}$$

where x is a vector of binary variables such that $x_j = 1$ if item j is selected, and $x_j = 0$ otherwise. The 0-1 MKP can be accepted as a special case of the general linear 0-1 integer programming problem with nonnegative coefficients. In the literature it is assumed that profits, weights and capacities are positive integers. However they can be easily extended to the case of real values (Martello & Toth, 1990).

The MKP is an NP-hard problem (Garey & Johnson, 1979) and the number of constraints increases its difficulty. Although the classical KP is weakly NP-hard, the MKP is much more difficult even for m = 2. According to Wilbaut, et al. (2008), the 0-1 MKP instances with 500 variables and 30 constraints cannot be solved optimally within a reasonable amount of computing time and memory requirement.

CHAPTER 3

MEMORY AND LEARNING IN METAHEURISTICS

One of the most important effects of improvement in modern sciences and technologies is understanding and modeling real life problems realistically and in more detail. The natural outcome of this fact is the rapid increase of problem dimensions and complexity, which challenges us to develop more sophisticated approaches. A powerful answer to this challenge can be based on solving problems by incorporating intelligence in the proposed solution methods. Intelligence can be defined as the ability to make the right decisions given a set of inputs and a variety of possible actions. In the problem solving arena, this is transformed into the term "artificial intelligence", or AI, that emerges by systematizing intellectual tasks relevant to human intellectual activity. AI employs intelligent procedures to understand and to create intelligent entities (Yang, 2010).

Computational Intelligence (CI) is a modern name for the subfield of AI (also named scruffy or soft) techniques. CI has a similar meaning to the well-known phrase AI, although CI is perceived more as a "bottom up" approach from which intelligent behavior can emerge, whereas AI tends to be studied from the *top down*, and derive from pondering upon the *meaning of intelligence* (Mumford & Jain, 2009). CI involves approaches based on strategy and outcome, and includes adaptive and intelligence systems, e.g. evolutionary computation, swarm intelligence (particle swarm and ant colony optimization) (Engelbrecht, 2007; Pedrycz, 1997).

Metaheuristics can be viewed as another name for the strategy-outcome perspective of scruffy AI. Metaheuristics or "modern heuristics" confront this challenge by adding strategies and mechanisms to existing construction and local search mechanisms in heuristics to avoid local optima (Moraga, 2009). A procedure in a metaheuristic is considered black box in that little (if any) prior knowledge needs to be known about it by the metaheuristic, and as such it may be replaced with a different procedure.

3.1 Concepts of Memory and Learning

There are substantial relationships between the term intelligence and the terms memory and learning. *Intelligence* is the ability that requires information captured by *learning* and stored in *memory* to make correct decisions in solving problems. The level of intelligence depends on the efficiency of learning activities and the capacity of memory; thus enhancing intelligence will then mean enhancing both memory and learning. Most researchers accept intelligence as an umbrella that covers the intellectual activities.

Webster's Dictionary (1996) defines memory as "the act or fact of retaining and recalling impressions, facts, etc."; and learning as "knowledge acquired by systematic study or by trial and error". Based on these definitions, we can see that the concepts of learning and memory are closely related. Furthermore, learning can be thought of as the acquisition of skill or knowledge, while memory as the expression of what you have acquired. Another factor that can be used in defining these two concepts is the rate at which the two processes occur: If the new skill or knowledge is gained slowly, that is

considered learning, but if the gain happens instantly, it is then considered creating memory (Kazdin, 2000).

The structure of memory is central to one's knowledge of the past, interpretation of the present, and prediction of the future (Kesner, 1998). Memory related to the past can be employed to create predictive models in the present, and therefore can guide current thoughts, decisions, and actions. Learning lets human beings have a greater degree of flexibility and adaptability than any other species.

Due to significant advancement in neuroscience, the concepts of memory and learning have undergone enormous changes over the last decade. In cognitive psychology, types of memory can be classified in a number of ways, depending on the criterion used. With duration as the criterion, it is divided into three functions for storage: sensory, short-term, and long-term (Anderson, 2000). Sensory memory takes the information provided by the senses and retains it accurately but very briefly. It is often considered part of the process of perception, and essential for storing information in short-term memory. The short-term memory temporarily records the succession of events, and determines what information moves from sensory memory to short-term memory. This information will quickly disappear forever unless we make a conscious effort to retain it. Sensory memory is a necessary step for short-term memory, and short-term memory is a necessary step toward the next stage of retention, long-term memory. Long-term memory is relatively permanent storage with information stored on the basis of meaning and importance. According to Anderson (2000), its capacity seems unlimited; however it sometimes distorts the facts, and tends to become less reliable as time goes by.

Based on the distinctions related to memory structure, learning can be accepted as a long-term change in mental representations or associations as a result of experience (Ormrod, 2008). If learning is a change in behavior, it can then be measured by observing the changes in behavior. The most common ways of measuring learning are recording the reduction in errors, the changes in the form and/or intensity of the behavior, the change in the speed with which a behavior is performed, and the change in the rate or frequency at which a behavior occurs (Chance, 2008).

Since memory stores and retrieves the information that is learned, it is then an essential component to all learning activities. Memory is nothing more than the record left by a learning process, and thus, memory depends on learning. But learning also depends on memory because the knowledge stored in memory provides the framework to new knowledge.

3.2 Memory and Learning in Metaheuristics

Alan Turing, known as the founder of artificial intelligence, was probably the first to use heuristic algorithms during the Second World War in breaking German Enigma ciphers via his cryptanalytic electromechanical machine, the Bombe. The Bombe used an algorithm to search for the correct setting coded in an Enigma message among about 10^{22} potential combinations. Turing named his search method as heuristic search, as was expected to work most of the time, but there was no guarantee to find the correct solution; it was a great success, nevertheless (Yang, 2010).

The mechanisms of memory and learning in algorithms store various information related to search history so that the algorithm can reach high quality solutions. Learning

takes place when the problem at hand is not well known at the beginning, and its structure becomes clearer and clearer when more experience with the problem is gained. Online learning is the type of learning in which an algorithm uses task-dependent local properties for a problem instance while it is solving that instance to determine the appropriate level trade-off between diversification and intensification (Yang, 2010). Different memory and learning structures have been used in different metaheuristics, as shown in Table 2 in which only Tabu Search (TS) is a single-solution metaheuristic and the rest are population-based metaheuristics.

Table 2. Memory Structures in Some Metaheuristics (Adapted from Talbi, 2009)

Metaheuristics	**Search Memory**
Tabu search	Tabu list
Evolutionary algorithms	Population of individuals
Scatter search	Population of solutions
Path relinking	Population of solutions
Ant colony optimization	Pheromone matrix
Particle swarm optimization	Population of particles
Estimation of distribution algorithms	Probabilistic learning model

The memory and learning structures can be described in the best way by taking TS algorithm as a baseline. In the memory and learning structures of TS, four main aspects are defined; recency, frequency, quality, and influence. The recency-based

memory keeps track of the attributes of the solutions found in the search process which have changed in the recent past. Attributes found in the solutions visited recently are defined as tabu-active which are called tabu in TS.

While the aspect of recency can be accepted as a short term memory implementation, the aspect of frequency deals with the long term TS strategies. The frequency-based memory consists of mainly two ratios: transition frequencies, which record how often the attributes are changed, and residence frequencies, which record how often the attributes are component of solutions produced. In scheduling for example, the number of times job *j* has been moved to an earlier position in the sequence can be an example for transition frequencies, and the sum of tardiness of job *j* when it occupies position P_j can be an example for residence frequencies (Glover & Laguna, 1997). The quality-based memory discovers the common elements in good solutions, or the paths that lead to good solutions. In these mechanisms some penalties can also be applied to move away from poor solutions. The last aspect of influence-based memory considers the effects of the decisions made in the solution process on both the quality and the structure. The quality aspect can be accepted as a special case of the influence aspect.

Intensification and diversification are two important strategies for the memory structure. According to Rochat and Taillard (1995), "diversification drives the search to examine new regions, and intensification focuses more intently on regions previously found to be good" Intensification strategies modify the algorithm to search the promising regions more thoroughly based on high quality solution features found in the search process, or by modifying choice rules to favor the inclusion of attributes of these solutions. These strategies focus on inspecting the neighborhood of elite solutions by

incorporating their good attributes into new solutions. On the other hand, diversification strategies encourage the algorithm to explore new regions and mainly utilize long term memory mechanisms. Local search optimization methods often rely on diversification strategies to reach better solutions. Diversification strategies help prevent cycling of the search process, and give more robustness to the algorithm.

The more sophisticated version of a tabu search includes longer term memory with associated intensification and diversification strategies. Glover and Laguna (1997) define this approach as Adaptive Memory Programming (AMP) because it is based on exploiting the strategic memory components. Taillard, et al. (2001) sketch the following algorithm of AMP based on the common features of the methods that use these strategic memory components:

1. Initialize memory.
2. Until the stopping criteria are met, do:
    a. Generate a temporary solution *s* using data stored in the memory;
    b. Improve *s* by implementing local search, *s'*; and
    c. Update the memory using data brought by *s'*.

Based on the AMP approach, Dréo et al. (2007) present Adaptive Learning Search (ALS) emphasizing that the memorized data are not only raw input, but provide information on the distribution and, thus, on the solutions. The algorithm for ALS consists of the following steps:

1. Initialize a sample.
2. Until the stopping criteria is met, do:
    a. Sampling: either explicit, implicit or direct;

b. Learning: the algorithm extracts information from the sample;

c. Diversification: it searches for new solutions;

d. Intensification: it searches to improve the existing sample; and

e. Replace the previous sample with the new one.

The main difficulty for metaheuristic search is the issue of balancing the intensification and diversification strategies. The search process can easily converge toward a local optimum and to diversify the search process, or to visit the solutions with different attributes, requires increasing the number of moves or components that are labeled as undesirable. For TS, the discussion then turns into finding the optimum tabu list size. Indeed, the reactive TS is designed to automatically adapt the tabu list size (Battiti & Tecchiolli, 1994).

The term reactive search supports the integration of learning techniques into metaheuristic search to solve complex optimization problems. The word reactive here describes an immediate response to events during the search through an internal feedback loop for online adaptation. The knowledge related to the search history is utilized for adaptation in an automatic manner. The algorithm keeps the ability to respond to different situations during the search process, but the adaptation is automated, and executed while the algorithm runs on a single instance reflecting on its past experience. Intelligent optimization refers to a more extended area of research, including online and offline schemes based on the use of memory, adaptation, and incremental development of models, experimental algorithmics applied to optimization, intelligent tuning, and design of metaheuristics (Battiti, Brunato & Mascia, 2008).

3.3 Metaheuristics with Memory and Learning

Memory and learning in metaheuristics represent the information extracted and stored during the search for better solutions. The content of these mechanisms varies from a metaheuristic to another (Table 2). While tabu list represents memory in TS, in most of the metaheuristics such as evolutionary algorithms and scatter search, the search memory is limited to the population of solutions. In Ant Colonies Optimization (ACO), the pheromone matrix is the main component of the search memory, whereas in Estimation Distribution Algorithms, it is a probabilistic learning model that composes the search memory.

Of the algorithms in Table 2, Estimation Distribution Algorithms and Path Relinking proposed to be incorporated into Meta-RaPS will be discussed in detail in the following chapters with Q learning algorithm from machine learning area and algorithms with adaptive parameter tuning.

3.3.1 Tabu Search

Tabu Search (TS) algorithms, introduced by Glover (1989), are one of the most common single-solution based metaheuristics that improve a single solution. The major property of this approach emerges from storing information related to the search process, which is called memory. A TS can be classified either as A|N|1 or A|N|P. The reason behind this classification is that TS employs adaptive memory using a neighborhood search and it moves from one current solution to the next after every iteration.

A TS begins with local or neighborhood search and generally the whole neighborhood is explored deterministically and the best solution found in the

neighborhood is selected as the new current solution. According to Talbi (2009), a TS may be considered as a dynamic transformation of the neighborhood; however, this mechanism may create cycles, which in order to be avoided, the TS "memorizes" the recent search trajectory by means of a tabu list. Usually, a tabu list consists of a constant number of solutions or attributes of the moves, which are updated at each iteration of the search process. Besides the tabu list, there is another mechanism called the aspiration criteria, to accept a solution that is "good" even though it is in the tabu list. A common aspiration criterion is if a solution is better than the best solution so far.

Due to the fact that a tabu list generally contains the information of recent solutions or moves, it can be classified as a short-term memory. Along with the short-term memory, in a TS there are medium-term and long-term memory mechanisms to apply for different purposes. While the medium-term memory, or intensification memory, stores the elite solutions and gives priorities to their attributes, the long-term or diversification memory, keeps the information of the visited solutions to use in exploring unvisited regions in the solution space.

3.3.1.1 Reactive Tabu Search

As a design parameter, the size of the tabu list plays a very important role in reaching high-quality solutions. Increasing the size of the tabu list can prevent cycles; however it can constrain the search process in a certain region, too. To handle this trade-off, various methods are developed in the literature. During the search process, the robust tabu approach chooses randomly different tabu list sizes from a specified range, and the deterministic tabu approach picks tabu list sizes that are previously assigned. A common

feature of these methods is that they require a fixed range determined before the start of the search process (Wassan, 2007). These facts brought Battiti and Tecchiolli (1994) to the more sophisticated version of the TS, a reactive tabu search in which the size of the tabu list dynamically, or reactively, adapts as the search progresses. They created an analogy between the evolution of the search process in combinatorial optimization and the theory of dynamic systems. According to the authors, similar to a dynamic system, three cases should be avoided in the search process: local minima, limit cycles, and chaotic attractors. Local minima are attractors of the system dynamics, and they are fixed points until the system is enforced by some phenomena to leave the local optimum and continue the search process. Limit cycles, or closed orbits, denote the case of visiting solutions previously found in the search process. Even in the absence of local minima and limit cycles, the solution space can be narrowed or deformed, and the search process can visit only parts of the solution space due to the chaotic attractors (Chiang & Russell, 1997). Battiti and Tecchiolli (1994) used the term *chaotic attractor* as an example of a dynamic behavior that could affect the search process. In their study, chaotic attractors are identified "by a contraction of the areas, so that trajectories starting with different initial conditions will be compressed in a limited part of the solution space, and by a sensitive dependence upon the initial conditions, so that different trajectories will diverge". They suggested that for an effective and efficient search process, preventing limit cycles is not enough, and the chaotic-like attractors should be removed too.

According to Glover and Laguna (1993), avoiding cycles is not the ultimate purpose of the search process; another purpose is to continue the exploration of new solution regions. To reach these goals, reactive tabu search implements two mechanisms:

first is adapting the size of tabu list (tabu tenure) throughout the search process depending on the repetitions of the solutions. The algorithm stores the information related to the solutions visited during the search process to control the repetitions and the interval between visits. The mechanism increases the size of tabu list when the number of repetitions exceeds a certain threshold, and vice versa. The second mechanism is an escape or diversification strategy, to take the search process out from its current region randomly if it repeats itself excessively (Wassan, 2006), or in other words, when there is evidence for chaotic attractors in the search space.

While adapting the size of the tabu list, intensification strategies are also employed to deeply search the area that gives good or elite solutions. Reactive tabu search algorithms aim to balance the intensification and diversification functions to control and run the search process fluently. As in the basic tabu search, in addition to the tabu list, the aspiration criteria also help prevent getting trapped at a local optimal solution.

3.3.2 Evolutionary Algorithms

The works of Mendel on the heredity from parents to offspring, and Darwin's theory of evolution presented in his famous book *On the Origin of Species* from the 19th century have inspired computer scientists in designing evolutionary algorithms (EAs) in the 1980s. Since then different approaches have evolved independently in the evolutionary algorithms area: Genetic algorithms, mainly developed by Holland (1962; 1975); evolution strategies, developed by Rechenberg (1965; 1973) and Schwefel (1965);

evolutionary programming by Fogel (1962; 1966) and genetic programming proposed by Koza (1992). Each of these approaches is inspired by the principles of natural evolution.

Genetic Algorithms (GA) are generally associated with binary representations; however, other types of representations can also be employed in different versions of GAs. The GA usually implements a crossover operator to two solutions having a "good" fitness values, and a mutation operator to modify the individual solution to create diversity. The replacement, or survivor selection, is performed by replacing the parents systematically with offspring. The basic crossover operator is based on a n-point or uniform crossover while the mutation is bit flipping. Probabilities are applied to both of the crossover and mutation operators.

Evolution Strategies (ES) are mostly applied to continuous optimization where the problem representations are based on real-valued vectors. ES usually use an elitist replacement strategy, and a normally (Gaussian) distributed mutation, while crossover is rarely used. An individual is composed of the problem's decision variables as well as some search parameters in order to evolve both the solution and the strategy parameters (e.g., mutation step size) at the same time. Their main advantage is their efficiency in terms of time complexity (Talbi, 2009).

Evolutionary programming (EP) mainly uses mutation, but not recombination or crossover. Traditional EP algorithms have been developed to evolve finite state machines to solve time series prediction problems and more generally to evolve learning machines (Fogel, 1966). Contemporary EP algorithms have later been applied to solving continuous optimization problems using real-valued representations. They use normally distributed mutations and self-adaptation principle of the parameters as in ESs. The

parent selection operator is deterministic, while the replacement operator is probabilistic and is based on a stochastic tournament selection (Eiben, 2003). EP is less used than the other approaches of EAs because of its similarity to ES.

Genetic programming (GP) expands the scope of the generic model of learning to the space of programs. Its main distinction from other EAs approaches is that the evolving individuals are themselves programs (nonlinear representation based on trees) instead of fixed length strings from a limited alphabet of symbols (linear representation). In GP, the parent selection is based on fitness proportions and the survivor selection is a generational replacement. The crossover operator is based on subtrees exchange and the mutation is based on random change in the tree. One of the main problems in GP is the uncontrolled growth of trees, which is called bloat. Theory of GP is less developed than in evolution strategies and genetic algorithms (Langdon & Poli, 2002) and it is widely applied in machine learning and data mining tasks such as prediction and classification.

In EAs, the population is usually generated randomly. Every individual in the population is an encoded version of a solution that is called *chromosome* while the decision variables within a solution (chromosome) are genes. The possible values of variables (genes) are the alleles and the position of an element (gene) within a chromosome is called locus. An objective function stands for a fitness value which shows the ability of an individual or a solution to survive in its environment. At each step, individuals are selected to form parents depending on their fitness value; individuals with better fitness are selected with a higher probability. The selection mechanism will lead the population to better solutions. However, individuals not having "good" fitness are not discarded immediately since they may have useful genetic material for future operations.

The selection process is executed by assigning a strategy, e.g. roulette wheel selection, tournament selection, stochastic universal sampling, or rank-based selection.

The selected individuals are then reproduced using variation operators (e.g., crossover, mutation) to generate new offspring. Finally, a replacement mechanism is applied to select which individuals (parents and offspring) of the population will survive to the new generation. Mutation operators are unary operators acting on a single individual representing small changes to selected individuals of the population. The probability P_m defines the mutation probability for each element (gene) of the representation. In general, small values are recommended for this probability ($P_m \in [0.001, 0.01]$). Some strategies initialize the mutation probability to 1/k where k is the number of decision variables, meaning that only one variable is mutated. The role of crossover operators is to pass down some characteristics of the two parents to generate the offspring. Unlike unary operators such as mutation, the crossover operator is binary and sometimes *n*-ary. The crossover probability P_c represents the proportion of parents on which a crossover operator will act. Common values for crossover probability are typically selected in the interval [0.45, 0.95].

The population size is another important parameter for EAs and usually larger population sizes have greater chances of converging to better or optimal solutions. While the sampling errors become more important in smaller populations, the time complexity of EAs grows linearly with the size of the population. A proper level of population size between the quality of the obtained solutions and the search time must be determined. In practice, a population size between 20 and 100 is usually considered typical.

3.3.3 Scatter Search

The concept of scatter search (SS), first proposed by Glover (1977), is a deterministic algorithm applied to both combinatorial and continuous optimization problems. SS is a population metaheuristic that recombines solutions selected from a reference set to build others, and from this point of view, it can be seen as an evolutionary algorithm (Glover, Laguna & Marti, 2003). SS creates the reference set by selecting "good" solutions from the population obtained in the previous search process. The selected solutions from the reference are combined to provide starting solutions to an improvement procedure, and the reference set is updated to incorporate both high-quality and diversified solutions. The diversity can be measured by taking the minimum Hamming distance from a solution to any solution selected for the reference set. The set of solutions is evolved by the use of recombination of solutions and applying some local search algorithms.

SS is designed by integrating of five methods:

- A Diversification Generation Method to generate a set of diverse initial solutions in order to diversify the search by selecting high-quality solutions.
- An Improvement Method to transform a trial solution into one or more enhanced trial solutions, in general, by applying a local search procedure.
- A Reference Set Update Method to create a reference set from the "best" solutions by keeping both diverse and high-quality solutions.
- A Subset Generation Method to operate on the reference set, to produce a subset of its solutions as a basis for creating combined solutions. This method is similar to the selection operator in EAs with the differences being, first, the

SS uses a deterministic operator, whereas in EAs, it is generally a stochastic operator; second, the size of the reference set in SS is much smaller than the size of the population in EAs (Talbi, 2009).

- A Solution Combination Method to transform a given subset of solutions produced by the Subset Generation Method into one or more combined solutions. The combination method can be seen as the crossover operator in EAs where more than two individuals are recombined.

3.3.4 Swarm Intelligence

In the field of optimization there are some promising algorithms inspired by the behavior of some species such as ants, birds, fish, bees, etc. These types of algorithms are called swarm intelligence algorithms. The expression "swarm intelligence" was first used by Beni, Hackwood, and Wang (Beni, 1988; Beni & Wang, 1989; Hackwood & Beni, 1992) in the context of cellular robotic systems. Swarm intelligence is defined as a field of computer science which is focused on the efficient computational methods for solving problems in a way that is inspired by the behavior of real swarms or insect colonies (Bonabeau, Dorigo & Theraulaz, 1999: Kennedy, Eberhart & Shi, 2001). The main characteristics of (artificial) swarm intelligence algorithms are that the particles, or species, are simple and nonsophisticated agents; they cooperate by an indirect communication instrument; and they move in the decision space of the optimization problem (Ahuja, et al., 2002).

Indeed, the behavior of real species is complex; they can process a lot of sensory inputs, which means a large amount of information. However, the complexity of the

species is still not sufficient to describe what these social colonies can do. This issue of how to connect individual behavior with collective performance can be explained by using self-organization (SO) concept, and in reality, the activities of social species are self-organized. SO theories originally developed in the context of physics and chemistry but have been extended to social insects to show that complex collective behavior may emerge from interactions among individuals that exhibit simple behavior (Haken, 1983; Nicolis & Prigogine, 1977). Recent research shows that SO is a major component of a wide range of collective phenomena in social species (Deneubourg, et al., 1989). The modeling of social species by means of SO can help design artificial distributed problem-solving devices that self-organize to solve problems, or in other words swarm-intelligent systems. SO is based on four elements (Bonabeau, Dorigo & Theraulaz, 1999).

- Positive feedback (amplification) promotes the creation of structures. For instance, recruitment to a food source is a positive feedback that relies on trail laying and trail following in some species like ants.
- Negative feedback counterbalances positive feedback and helps stabilize the collective pattern; it may take the form of saturation, exhaustion, or competition.
- Amplification of fluctuations (random walks, errors, random task-switching, etc.). Not only do structures emerge despite randomness, but randomness is often crucial since it enables the discovery of new solutions, and fluctuations can act as seeds from which structures nucleate and grow.
- Multiple interactions. A single individual can generate a self-organized structure, however, SO generally requires a minimal density of mutually tolerant

individuals. Moreover, individuals should be able to make use of the results of their own activities as well as of others'.

SO in social insects often requires interactions among insects and such interactions can be direct or indirect. Indirect interactions are more subtle however; two individuals interact indirectly when one of them modifies the environment and the other responds to the new environment at a later time. This type of interaction is an example of stigmergy, which was introduced by Grasse (1959; 1984) and is considered the second most important theoretical concept of swarm intelligence after self-organization. Stigmergy (from the Greek stigma: sting, and ergon: work) does not describe how species coordinate their activities, however, it does provide a general mechanism that relates individual and colony-level behaviors: individual behavior modifies the environment, which in turn modifies the behavior of other individuals.

The most successful swarm intelligence inspired optimization algorithms are ant colony and particle swarm optimization. Besides the wide range of applications of swarm intelligence in the literature, hybrid techniques in which swarm intelligence algorithms work with other metaheuristics can also be a promising concept to make use of both the intelligence of swarms and the efficiency of metaheuristics.

3.3.5 Ant Colony Optimization

Ant colony optimization (ACO) is one of the most successful swarm intelligence algorithms. The possibility of "forming of communication by means of modifications of the environment" is defined as stigmergy, which is one of the basic concepts for the ACO (Dréo et al, 2006).

The ACO aims to imitate the real ants as multiagent systems to solve optimization problems and was first proposed by Dorigo (1992). Even though real ants cannot see well, they can find the shortest path between two points. In this process they are using a very simple and yet powerful mechanism; a chemical trail called a pheromone. The ants follow their routes according to the amount of pheromone; the larger the amount of the pheromone on a route, the larger the probability of being selected by the ants. However the pheromone is a volatile substance and it decreases over time. In the beginning of the process, the probabilities of selecting the routes by ants are equal, but since the shorter routes need less time to travel, they will emerge with higher rates of selection due to higher amounts of pheromone. This process, supported by the evaporation mechanism, will end up with finding the shortest path. The pheromone trail, in essence, represents the long term memory of the entire system and where information related to the process is stored (Dorigo & Stützle, 2004).

ACO is composed of two main steps: construction of solutions and updating the pheromone. In the first step solutions are constructed by adding solution components to partial solutions according to the probabilistic transition rule in equation (3.1):

$$P_{ij}^{k} = \frac{[\tau_{ij}]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{l \in N_i^k} [\tau_{il}]^{\alpha}[\eta_{il}]^{\beta}}. \tag{3.1}$$

where τ_{ij} is pheromone desirability, η_{ij} is heuristic desirability, α is ratio of pheromone desirability ($0 < \alpha < 1$), and β is ratio of heuristic desirability ($0 < \beta < 1$) for selecting component *j* after the component *i*. By using this probabilistic transition the construction algorithm takes into account both the amount of pheromone and problem-dependent heuristic information.

In the second step the amount of pheromone is updated in two phases: evaporation phase and reinforcement phase. In the evaporation phase the pheromone trail is reduced by a fixed ratio q ($0 < q \leq 1$) for all components in the decision space by applying equation (3.2). This evaporation process protects all ants from a premature convergence toward good solutions and encourages diversifying the search space.

$$\tau_{ij} = (1 - q)\, \tau_{ij} \tag{3.2}$$

In the reinforcement phase, the amount of the pheromone is updated according to solutions generated by using two main strategies: online and offline updates. In the case of online updating, the pheromone trail is updated by an ant either at each step of the solution construction (step-by-step updating) or after a complete solution is generated (delayed updating). The offline updating is more popular where the updating process is applied only after all ants generate a complete solution. In this approach different strategies can be performed including quality-dependent, rank-based, elitist solution, best-worst, moving average, and minimum pheromone values update (Merkle & Middendorf, 2005).

The selection of the of ACO parameters plays a critical role in the search process. Therefore, a good trade-off between the ratios of the pheromone desirability (or intensity), and heuristic desirability (or visibility) must be found to balance intensification and diversification. If the ratio of pheromone desirability is equal to 0, the ACO algorithm will act like a stochastic greedy algorithm, and if the ratio of heuristic desirability is equal to 0, only the pheromone trails will guide the search.

ACO can be classified as a construction and population-based metaheuristic, which although has been created mainly to solve discrete optimization problems, it has been extended to deal with continuous optimization problems.

3.3.6 Particle Swarm Optimization

Particle swarm optimization (PSO) is a stochastic population-based metaheuristic inspired by swarm intelligence. PSO simulates the social behavior of natural organisms, e.g. bird flocking or fish schooling, in search of food. Among these organisms, or the swarm, a dynamic behavior in relatively complex displacements can be observed, where the individuals have access to limited information, like their closest neighbors' positions and speed (Dréo, et al, 2006). Each individual uses the local information regarding this displacement to decide on its own displacement. In other words, a coordinated behavior using local movements emerges without any central control.

In PSO algorithms, each individual particle of a swarm represents a potential solution in a multidimensional search space. The particles start searching randomly for the optimal solution of a given objective function by moving through the search space. The objective function measures the quality or amount of food at each place and the particle swarm searches for the place with the best or most food (Merkle & Middendorf, 2005). The position of each particle is adjusted according to its velocity (i.e., rate of change) and the difference between its current positions, the best position found by its neighbors, and the best position it has found so far. As the model is iterated, the swarm focuses more and more on an area of the search space containing high-quality solutions (Blum & Li, 2008).

The individual particle is represented by the vector x_i, which has its own position and velocity. Each particle adjusts its position according to the global optimum with respect to two factors: the best position visited by itself (p_{besti}) denoted by the vector p_i, and the best position visited by the whole swarm (g_{best}) denoted by the vector g_i. The vector (p_i - x_i) shows the difference between the current position of the particle *i* and the best position of its neighborhood. The neighborhood, which must be defined for each particle, describes the social influence between the particles in the swarm. To define a neighborhood, two methods are traditionally used: the global best method and the local best method. In the global best method, the neighborhood is defined as the whole population of particles, whereas in the local best method, the neighborhood of a particle is the set of directly connected particles, in which case, the neighborhood may be empty and the particles isolated. A particle is composed of three vectors: the x-vector for its current position, the p-vector for the location of the best solution found so far by the particle and the v-vector for the direction of the particle to travel in the search space. In each iteration, the movement of the particle can be given by equation (3.3):

$$x_i(t) = x_i(t-1) + v_i(t) \tag{3.3}$$

Updating of the particles' positions is dependent on the direction of their movement, their speed, the best preceding position p_i and the best position p_g among the neighbors as shown in the equation (3.4):

$$v_i(t) = v_i(t-1) + \rho_1\alpha_1 \times (p_i - x_i(t-1)) + \rho_2\,\alpha_2 \times (p_g - x_i(t-1)) \tag{3.4}$$

where ρ_1 and ρ_2 are random variables in the range [0, 1], and α_1 and α_2 represent the learning factors. The parameter α_1 is the cognitive learning factor that decides the level that a particle has toward its own success, and the parameter α_2 is the social learning

factor that reflects the level of attraction that a particle has toward the success of its neighbors. Socio-psychology suggests that the movements of the individuals are influenced by their last behavior and that of their neighbors who are closely placed in the social network and not necessarily in space.

To control the balance between intensification and diversification of the search space, a weight *w*, called inertia, is generally added to the velocity update procedure, as in equation (3.5):

$$v_i(t) = w \times v_i(t-1) + \rho_1 \times (p_i - x_i(t-1)) + \rho_2 \times (p_g - x_i(t-1)) \quad (3.5)$$

A large inertia weight encourages diversification of the search, and a smaller inertia weight encourages intensification of the search in the current region. According to the new velocity, each particle updates its position in the solution space was given in equation (3.3).

After these updates each particle will update the best local solution, $p_i = x_i$ if $(x_i) < p_{besti}$, and the best global solution of the swarm, $g_i = x_i$ if $(x_i) < g_{best}$. As such, a particle changes its position after each iteration according to its own and to its neighbors' positions.

Unlike ACO algorithms, PSO has been successfully designed originally for continuous optimization problems; however, by employing velocity models, PSO can be applied to discrete optimization problems also. Velocity models for discrete optimization problems are inspired from mutation and crossover operators in EAs. The velocity models may be real valued, stochastic, or based on a list of moves. In stochastic velocity models for binary encodings, the velocity is associated with the probability for each binary dimension to take value of 1.

CHAPTER 4

METHODOLOGY

In the process of incorporating memory and learning mechanisms into Meta-RaPS, four of the proposed approaches, EDA, Q learning, Path Relinking, and adaptive parameter tuning, will be investigated throughly and designed to create its own main version of Meta-RaPS. For each of these main versions, two versions will be introduced depending on how solutions are generated for their memory mechanisms: randomly (or simply using the weights of items) and using a greedy rule. Although the first type is not expected to produce high quality solutions, it may give an idea about the pure contribution of memory and learning ability to an independent algorithm, i. e. not getting any help from a greedy rule. And the main idea for the second type is to guarantee obtaining high quality solutions by applying a greedy rule.

After completing these four proposed algorithms, the Meta-RaPS was redesigned as the fifth algorithm by utilizing all the lessons learned from the efforts of incorporating the memory and learning mechanisms. All these proposed algorithms were evaluated and reported by following the same method.

4.1 Performance Comparison of the Proposed Algorithms

Due to the existence of strong randomness component in the proposed Meta-RaPS versions were run 10 times for each instance and the average will be taken for all runs. After completing the solution process, the performance of each algorithm will be reported in terms of solution quality, or percentage deviation, number of iterations, CPU time, and

frequency of reaching optimum/best solutions. The percentage deviations were calculated using equation (4.1):

$$\frac{f(s^*)-f(s)}{f(s^*)} x100 \qquad (4.1)$$

where s is the solution found in the current method and s^* is the optimum solution/best solution. The percentage deviations were calculated not only for the average of best improved solutions (IMean), but also for the best of the best improved solutions found in 10 runs (IBest) and for the average of constructed solutions (CMean). While IMean shows the mean performance of the algorithm, IBest may give an idea about the limits of the algorithm. CMean helped evaluate the quality of the initial solutions produced by using the priorities of memory and learning mechanisms since the improvement phase is same for all algorithms.

In terms of the frequency of reaching optimum/best solutions, the number of times the algorithm has found the optimum/best solutions in 10 runs were given under the heading of Optimum Frequency. The heading "Optimum Instance" shows the number of instances solved optimally/with best solutions by the algorithm. To finish the report, the averages and standard deviations for these metrics were calculated.

Comparison of the proposed Meta-RaPS algorithms were implemented in the following aspects:

- Between versions of proposed Meta-RaPS algorithms,
- Before and after memory and learning inclusion into Meta-RaPS, and
- With other applications in the literature applied to solve 0-1 MKP.

For the second aspect, the solutions of the proposed algorithms were compared with the results of the original versions of Meta-RaPS by Moraga et al. (2005).

4.2 Stopping criteria

Stopping criteria for the proposed algorithms are:

- To run the algorithms 10,000 iterations for small, medium and large size instances,
- Or, to stop whenever the deviation% of the solution from the optimal/best found solution becomes 0, whichever comes first.

Although the small and medium size instances do not need 10,000 iterations for their solution process, this number of iterations is accepted only to be consistent with the number of iterations selected by Moraga, et al. (2005) in their Meta-RaPS approach.

4.3 0-1 MKP Instances

To test and compare the performance of the proposed algorithms, they will be applied in 0-1 MKP test instances in the literature. The standard library of 55 small and medium size 0-1 MKP test instances in the literature developed by Petersen (1967), Weingartner and Ness (1967), Shih (1979) and Fréville and Plateau (1990) were solved by proposed versions of Meta-RaPS. Details of small and medium size 0-1 MKP problems are shown in Table 3.

For large size 0-1 MKP test instances, 270 test instances generated by Chu and Beasley (1998) were used. These 0-1 MKP test instances are created by accepting the tightness ratios of 0.25, 0.50 and 0.75 for each group of 10 instances in the set, respectively. The tightness ratio α is defined as the ratio between the constraint value and the sum of the corresponding weights (4.2).

Table 3. Small and Medium Size 0-1 MKP Test Instances

#	Name	Item	Knapsack	#	Name	Item	Knapsack
1	HP1	28	4	29	WEISH04	30	5
2	HP2	35	4	30	WEISH05	30	5
3	PB1	27	4	31	WEISH06	40	5
4	PB2	34	4	32	WEISH07	40	5
5	PB4	29	2	33	WEISH08	40	5
6	PB5	20	10	34	WEISH09	40	5
7	PB6	40	30	35	WEISH10	50	5
8	PB7	37	30	36	WEISH11	50	5
9	PETERSEN1	10	6	37	WEISH12	50	5
10	PETERSEN2	10	10	38	WEISH13	50	5
11	PETERSEN3	15	10	39	WEISH14	60	5
12	PETERSEN4	20	10	40	WEISH15	60	5
13	PETERSEN5	28	10	41	WEISH16	60	5
14	PETERSEN6	39	5	42	WEISH17	60	5
15	PETERSEN7	50	5	43	WEISH18	70	5
16	SENTO1	60	30	44	WEISH19	70	5
17	SENTO2	60	30	45	WEISH20	70	5
18	WEING1	28	2	46	WEISH21	70	5
19	WEING2	28	2	47	WEISH22	80	5
20	WEING3	28	2	48	WEISH23	80	5
21	WEING4	28	2	49	WEISH24	80	5
22	WEING5	28	2	50	WEISH25	80	5
23	WEING6	28	2	51	WEISH26	90	5
24	WEING7	105	2	52	WEISH27	90	5
25	WEING8	105	2	53	WEISH28	90	5
26	WEISH01	30	5	54	WEISH29	90	5
27	WEISH02	30	5	55	WEISH30	90	5
28	WEISH03	30	5				

$$\alpha = \frac{b_i}{\sum_{j=1}^{n} a_{ij}} \in \{0.25, 0.5, 0.75\} \tag{4.2}$$

Resource consumptions a_{ij} are random numbers assigned between (0, 1,000), and profits are correlated to the weigths c_j are are generated via equation (4.3):

$$c_j = \sum_{i=1}^{m} \frac{a_{ij}}{m} + (500 r_i) \in \{0.25, 0.5, 0.75\} \tag{4.3}$$

where m is the number of knapsacks and r_i is a random number generated from (0, 1]. Using the gap, i. e. the relative distance between the best integer value found in the branch and bound and the LP value of the best unexplored node in the three (Osorio & Cuaya, 2005), as a measure of hardness, Pirkul (1987) concluded that the gap increases as the constraints become tighter, if the number of variables and constraints are constant. Distribution of large size 0-1 MKP problems in terms of item and knapsack is summarized in Table 4. Besides best/optimal solutions found for these test problems, the LP relaxation values for these large size test problems are also available in the OR-Library (Beasley, 1990).

Table 4. Large Size 0-1 MKP Test Instances

	Knapsack		
Item	**5**	**10**	**30**
100	30	30	30
250	30	30	30
500	30	30	30
Total	270		

Two versions of the four proposed algorithms will be first employed to solve the small and medium size instances. Depending on their performances, one of these two versions will be selected as its main version of Meta-RaPS to solve first three sets of large size instances presented in the first row in Table 4. In the case of redesigning Meta-RaPS, all small, medium, and large 0-1 MKP instances will be solved.

4.4 Tuning Parameters

The values of the parameters for the metaheuristics have significant impact on both the solution process and solution quality. To obtain the best results, the issue of "finding the best parameter setting" for metaheuristics becomes an optimization problem by itself. There is no universal set of parameter for a certain metaheuristic to be applied to different problems. In fact, for different problems, there are different optimal selections (Wolpert & Macready, 1997).

In setting the parameters of the algorithms, two main forms are defined: parameter tuning and parameter control (Eiben, Hinterding & Michalewicz, 1999). In parameter tuning, or offline tuning, parameters are set *a priori*. In the case of parameter control, or online tuning, initial values for the parameters are assigned and changed during the search process.

One-Factor-At-a-Time (OFAT) and Design of Experiments (DOE) approaches are offline parameter tuning methods used to find the best parameter setting in the literature (Daniel, 1994). Unlike DOE, OFAT neglects the interactions between the parameters that might change the whole solution process and quality. Particularly, in terms of the interactions, DOE methods are promising approaches and can be employed to tune the

parameters more effectively. There have been many studies to tune the parameters of metaheuristics by means of DOE (e.g, Fığlalı, et al., 2009; Kramer, Gloger and Goebels, 2007; Li et al., 2009).

4.4.1 Design of Experiments (DOE)

To finetune the Meta-RaPS parameters in the most effective way, DOE offers 2-level (2^k) full factorial, orthogonal array, central composite and D-optimal design methods. 2-Level (2^k) full factorial design generated by using the Yates algorithm (Box, Hunter & Hunter, 1978) is one of the most widely used DOE tools where k is the number of factors. One drawback of 2^k full factorial design is the rapid increase of the number of experiments as the number of factors increases. The fact that effects of 3 or higher interactions tend to be insignificant, and therefore may be ignored, bring us to a fractional factorial design type named orthogonal array (OA) design where only main factors and the 2-factor interactions are considered.

In 2^k full factorial and OA designs, it is assumed that the relationship between the 2-level factors is linear. It is possible to increase the number of levels to 3 to capture the nonlinearity, however, it would be a bit controversial and none of the rules for the 2-levels would apply in those designs. Also, this would not be the best candidate for continuous factors like parameters used in metaheuristics. A better approach to cope with the nonlinearity and continuous factors could be Response Surface Methods (RSM) using Central Composite Design (CCD) developed by Box and Wilson (Box & Wilson, 1951).

After implementing multiple DOE methods to a GA parameter setting, it was observed that, D-optimal design is the most effective DOE methods in terms of tuning

parameter settings (Arin, Rabadi & Unal, 2011). This study encourages the application of a D-optimal design in tuning the parameters of the Meta-RaPS' proposed versions.

4.4.2 D-Optimal Design

CCD is quite an efficient design especially due to adding the second-order nonlinearity; however, in some cases it may not be enough to understand the relationships between factors, and also, the number of experiments must be kept to an absolute minimum. If a design has an absolute minimum number of experiments, such design is called "saturated design". Saturated designs are constructed by applied D-optimality criterion. Creating a D-optimal design begins with the estimator of simple linear regression in equation (4.4):

$$\hat{Y} = b_0 + \sum b_i x_i . \tag{4.4}$$

where b_0 is the intercept, b_i are the slopes. If this equation is written in matrix form, we will have (4.5):

$$Y = XB + \varepsilon . \tag{4.5}$$

The set of design B can be estimated in the form given in (4.6) by applying the Least Square Regression method.

$$B = (X^T X)^{-1} X^T Y . \tag{4.6}$$

A statistical measure of accuracy of B is the variance-covariance matrix in (4.7):

$$V(B) = \sigma^2 (X^T X)^{-1} . \tag{4.7}$$

where σ^2 is the variance of the error. V(B) is a function of $(X^T X)^{-1}$ and to increase the accuracy, $(X^T X)^{-1}$ should be minimized. Statistically, minimizing $(X^T X)^{-1}$ is equal to

maximizing the determinant of (X^TX). "D" in the term "D-optimal" comes from the first letter of the word "determinant" where the D-optimal design seeks to maximize $|X^TX|$.

The minimum number of experiments for D-optimal design is calculated as $(n+1)(n+2)/2$ where n is number of factors. To obtain more accurate results, D-optimal designs can be augmented by adding more experiments. D-optimal design with k = 3 is created by augmenting the design by two experiments in Table 5.

Table 5. D-Optimal Design with k = 3

Experiment	A	B	C	AB	AC	BC	A^2	B^2	C^2
1	-1	-1	-1	1	1	1	1	1	1
2	-1	-1	1	1	-1	-1	1	1	1
3	-1	0	0	0	0	0	1	0	0
4	-1	1	-1	-1	1	-1	1	1	1
5	-1	1	1	-1	-1	1	1	1	1
6	0	-1	0	0	0	0	0	1	0
7	0	0	1	0	0	0	0	0	1
8	1	-1	-1	-1	-1	1	1	1	1
9	1	-1	1	-1	1	-1	1	1	1
10	1	1	-1	1	-1	-1	1	1	1
11	1	1	0	1	0	0	1	1	0
12	1	1	1	1	1	1	1	1	1

Besides the advantages mentioned before, if some experiments are infeasible, D-optimal designs can still be used by extracting these experiments from the design. Some of the interesting features of D-optimal designs, unlike the previous DOE methods, that

they are not orthogonal, and there are no degrees of freedom to test the accuracy of the model. There are some heuristics (Box & Drapper, 1974), and software (SAS Institute, 2007) available to come up with a design that maximizes $|X^TX|$.

In applying D-optimal design, the first step is coding the parameters (priority as factor A, restriction as factor B, improvement as factor C) in lower, medium, and upper levels (-1, 0, 1). In each experiment, the factors, or parameters, are set and run according to the first three columns in the design shown in Table 6. After the solutions from the experiments are obtained, the results are analyzed by implementing regression analysis and its mathematical model is derived. The model can then be solved by a linear programming solver, such as MS Excel, to minimize the objective function, which is percentage deviation from optimum/best solution found. The parameter setting by D-Optimal is found after coding back these findings to their real values.

By applying the D-optimal design the parameter settings are tuned for the first three Meta-RaPS algorithms presented in Table 6 for small/medium and large size 0-1 MKP instances.

Table 6. Parameters of the Proposed Meta-RaPS Algorithms for 0-1 MKP Instances

Parameter	**Values**	
	Small/Medium 0-1 MKP	**Large 0-1 MKP**
Priority (p)	0.4	0.5
Restriction (r)	0.2	0.5
Improvement (i)	0.1	0.1
Number of iterations (I)	10000	10000

These parameter settings were used only for the versions of Meta-RaPS EDA, Meta-RaPS Q and Meta-RaPS PR as offline parameter tuning. In the Meta-RaPS AP version, parameters were tuned online, i.e. adaptively changed with the solution process. In the redesigned version of Meta-RaPS, the parameters found adaptively in the Meta-RaPS AP were used.

4.5 Statistical Comparison

In the statistical comparison of the proposed algorithms, the first step is to determine whether there are any significant differences between the means of the Meta-RaPS versions in terms of percentage deviation and computational time. Since more than two algorithms are proposed, it is inappropriate to employ a t-test which compares the pairs. In this case, the way to evaluate whether or not the difference between the algorithms is statistically significant is using a one-way analysis of variance (ANOVA). The one-way ANOVA applies the F-test to determine whether there is a significant difference among treatment means. When the null hypothesis (H_0: "Means of Percentage Deviations/Time for Meta-RaPS Versions Are Equal") is rejected via the one-way ANOVA, this shows that some of the treatment or factor level means are different but does not identify which ones are different. To determine which specific algorithms differ from each other, a *post-hoc* Multiple Comparisons Test is needed. Tukey's multiple comparison test is one of several tests that can be used to determine which means amongst a set of means differ from the rest. Tukey's multiple comparison test is also called Tukey's HSD (honestly significant difference) test. The test compares the difference between each pair of means with appropriate adjustment for the multiple

testing. The Tukey multiple comparison test, like the t-test and ANOVA, assumes that the data from the different groups come from populations where the observations have a normal distribution and the standard deviation is the same for each group (Montgomery & Runger, 2003). In the one-way ANOVA analysis, the significance level is accepted as 0.05.

4.6 Conditions for the Comparison

All of the proposed Meta-RaPS algorithms are implemented in C++. The small/medium instances are solved on the Intel i5 CPU 2.27 GHz PC and the large instances are solved on the Intel(R) Xeon(R) CPU E5690 3.47 GHz workstation. Since there are many parts in the proposed algorithm in which randomness plays a very important role, all of the proposed algorithms solved each instance 10 times, and their mean and standard deviations were calculated for analysis.

The proposed algorithms were created by incorporating different memory and learning mechanisms into Meta-RaPS. In other words, the only difference among all the proposed algorithms is the learning and memory mechanisms, while the Meta-RaPS approach keeps its main structure the same. By following this approach, the efficiency of different memory and learning mechanisms were compared in a fair way as shown in the following chapters.

CHAPTER 5

INCORPORATING ESTIMATION OF DISTRIBUTION ALGORITHMS INTO META-RAPS

In demonstrating the contribution of memory and learning into metaheuristics, Estimation of Distribution Algorithms (EDA) is the first method incorporated as a memory and learning mechanism into the Meta-RaPS. EDA is a recent stochastic optimization technique that explores the space of candidate solutions by sampling an explicit probabilistic model constructed from promising solutions found so far (Hauschild & Pelikan, 2011). EDA estimates the probability distribution for each decision variable and with the help of this distribution it generates new solutions. These new solutions will then replace the old population according to given rules. This process iterates until termination criteria are met.

5.1 Literature Review

The term EDA was first introduced by Mühlenbein and Paaß (1996) in their seminal work. They revealed that selecting individuals (i.e. solutions) by means of the estimation of their probability distribution was one way to create more efficient Evolutionary Algorithms (EAs). Later, Mühlenbein, et al. (1999) identified the factorization of the probability distribution according to a probability model, as a practical method that permits the computation of estimations which have been the core of what mostly known as EDAs (Santana, 2005). Since then, the growing interest in EDAs constituted a discipline in evolutionary computation.

Usually, the probabilistic model as well as the learning and sampling methods employed in EDAs are static. Santana, et al. (2008) presented a general framework for introducing adaptation in EDAs. Gao and Culberson (2005) identified criteria that characterize the space complexity of two typical implementation schemes of EDAs – the factorized distribution algorithm and Bayesian network-based algorithm. Chen, et al. (2010) investigated the computational time complexity of a simple EDA, the univariate marginal distribution algorithm (UMDA), in order to gain more insight into EDAs complexity.

Santana (2005) proposed a new probability model based on what in statistical physics is known as the Kikuchi approximation. Shakya and McCall (2007) presented a Markov random field (MRF) approach to estimating and sampling the probability distribution in populations of solutions. Handa (2007) incorporated mutation operators: a bitwise mutation operator and a mutation operator into EDA in order to maintain diversities in populations. Lima, et al. (2011) investigated the relationship between the probabilistic models learned by the Bayesian optimization algorithm (BOA) and the underlying problem structure.

In the literature there are different EDA designs developed for continuous optimization (Bosman & Grahl, 2008; Ding et al., 2008; Miquélez, et al., 2007; Xiao, et al., 2009), dynamic optimization problems (Yuan, et al., 2008), clustering (Ahn & Ramakrishna, 2006; Qiang & Xin, 2005), non-separable problems (Agapie, 2010) and polygonal approximation problems which is important especially in the area of pattern recognition (Wang, et al., 2009). Besides multiobjective EDA applications (Martí, et al., 2011; Qingfu, et al., 2008; Zhang, et al., 2008), EDA created high quality solutions when

hybridized with algorithms such as Particle Swarm Optimization (Hongcheng, et al., 2011), memetic algorithms (Huang, et al., 2010), neural networks (Zhou & Wang, 2010) and variable neighborhood search (Santana, et al., 2008). Quadratic assignment problem (Zhang, et al., 2003), software testing (Sagarna & Lozano, 2005), robust airfoil optimization (Zhong, et al., 2008), nuclear reactor fuel management optimisation (Jiang, et al., 2006) and real-time video tracking (Patricio, et al., 2009) are applications of EDA.

EDAs have also been used with permutation type optimization problems including scheduling problems such as nurse scheduling (Uwe & Jingpeng, 2007), flowshop scheduling (Jarboui, Eddaly & Siarry, 2009), job shop scheduling (Zhang, 2011) and project scheduling (Wang & Fang, 2012). Chen, et al. (2010) produced guidelines for designing EDAs in solving single machine scheduling problems.

Extensive information about EDA can be found in Pelikan, Goldberg and Lobo (2002), Lozano, et al. (2006), and Sastry, et al. (2006). Very recently, Hauschild and Pelikan (2011) introduced a research on the introduction and survey of EDA.

5.2 Estimation of Distribution Algorithms

EDA is one of the recent optimization techniques that belongs to the class of the population-based metaheuristics. EDA is based on the idea that the probability distribution created from promising solutions would keep giving higher probability to high quality solutions and approach the optimum. These new solutions are then used in updating the probability distribution by replacing the old population in terms of some criteria such as the fitness function or diversity which can be defined as the measure of distinctness among the solutions. The important aspect in EDA is that the probability distribution should not perfectly represent the population of promising solutions, but

rather capture the features of candidate solutions that make them better than other candidate solutions (Hauschild & Pelikan, 2011). EDA have been specifically designed for black box optimization (BBO) problems in which objective functions are not given in a closed analytical form (Grahl, 2007). In a BBO, the structure of an optimization problem is hidden from the optimization process and the only information that can be exploited is a quality measure that is assigned to candidate solutions.

EDA is an outgrowth of EAs where statistical information is obtained from the population to form a new population and the Darwinian operators are replaced by probability distributions. However, the main difference between most EAs and EDAs is that the probability distribution used in EDAs to generate new candidate solutions is defined explicitly whereas the distribution in most EAs is defined implicitly (Hauschild & Pelikan, 2011).

The main step in EDA is estimating the probability distribution P(x) which assigns to items the probability of being selected in each position. If the optimization problem is represented by a bit vector, the distribution is represented by a single vector of n probability elements $P = (p_1, p_2, ..., p_n)$. Each element of this probability vector stands for the probability of being included in the solution, i.e. 1 if selected, 0 otherwise with probability of $1 - p_n$. Assuming that the population size is large enough to ensure reliable convergence, the EDA based on the probability vector provides an efficient and reliable approach to solving many optimization problems (Goldberg, 2002).

While creating new populations, EDA implements a probabilistic learning model that is used as memory. If the probabilistic learning model can capture the important

features of promising solutions and create new solutions based on these features, then the EDA should be able to quickly converge to the optimum (Mühlenbein & Mahnig, 1999).

The learning model is a key element in EDA, thus EDAs are usually classified by the type of learning model. Since the interactions between the decision variables are very important in learning models, EDA takes into account the level of variable interactions in the probabilistic model. The assumption that decision variables of the problem are independent will often prevent convergence to the optimum when their interactions are strong. In terms of the interactions, EDA can be classified as univariate, bivariate, and multivariate EDAs.

In the class of univariate EDAs, no interactions between the decision variables are considered in the generation of the probabilistic model. Mathematically, a univariate model decomposes the probability of a candidate solution $(X_1, X_2, \ldots, X_n)$ into the product of probabilities of individual variables as in (5.1):

$$p(X_1, X_2, \ldots, X_n) = \prod_{i=1}^{n} p(X_i) \tag{5.1}$$

where $p(X_i)$ is the probability of variable X_i, and $p(X_1, X_2, \ldots, X_n)$ is the probability of the candidate solution $(X_1, X_2, \ldots, X_n)$. One of the most known univariate EDAs is Population-Based Incremental Learning (PBIL) which is the first EDA strategy applied to solve optimization problems (Baluja, et al., 1994). On the contrary to the EDAs keeping a population of candidate solutions, incremental EDAs fully replace the population with the probabilistic model. In PBIL, after generating new solutions, the best solution or the set of best solutions, is selected to create the probability distribution of best solutions, $P_{best} = (p_{1best}, p_{2best}, \ldots, p_{nbest})$, which will be used to update the probability distribution of solutions, $P = (p_1, p_2, \ldots, p_n)$, by using the rule in equation (5.2) (Saéz, 2009):

$$p_i = (1 - \alpha)\, p_i + \alpha\, p_{ibest}. \tag{5.2}$$

where α is the learning factor. A smaller learning factor implies a diversifying search process and a higher learning factor means an intensifying search process. According to Saéz (2009), the mutation operator plays also an important role during the search process to guarantee convergence, avoiding local optima, and maintaining the diversity through the iterations. The mutation operator in PBIL algorithms can be applied at two levels: solution vector or probability matrix to maintain genetic diversity. Besides the genetic algorithm operators, local search algorithms can also be implemented in EDA to enhance the solution quality (Zhang, et al., 2006). Besides PBIL, the univariate marginal distribution algorithm (UMDA) (Mühlenbein & Paaß, 1996) and the compact genetic algorithm (cGA) (Harik, et al., 1997) are other univariate type of EDAs.

In the bivariate EDAs, or tree-based models, there are interactions between two decision variables and the conditional probability of a variable may only depend on the other variable. The mutual-information-maximizing input clustering (MIMIC) is in the class of bivariate EDAs, and uses a chain distribution to model interactions between variables (De Bonet, et al., 1997). In MIMIC, given a permutation of the n variables in a problem, $\pi = i_1, i_2, \ldots, i_n$, the probability distribution of $p(X_1, X_2, \ldots, X_n)$ is formed as in (5.3);

$$p_\pi(X) = p\left(X_{i_1} \mid X_{i_2}\right) p\left(X_{i_2} \mid X_{i_3}\right) \ldots p\left(X_{i_{n-1}} \mid X_{i_n}\right) p\left(X_{i_n}\right) \tag{5.3}$$

where $p\left(X_{i_j} \mid X_{i_{j+1}}\right)$ is the conditional probability of X_{i_j} given $X_{i_{j+1}}$. Candidate solutions are generated by sampling this probability distribution. To improve the expressiveness of the probabilistic models compared to MIMIC, Baluja and Davies (1997) used dependency trees to model promising solutions. The other bivariate EDA is the bivariate

marginal distribution algorithm (BMDA) that constructs a model based on a set of mutually independent trees (a forest) (Pelikan & Mühlenbein, 1999).

Multivariate EDAs define the probabilistic model considering the interactions among more than two decision variables. While univariate and bivariate models provide EDAs with the ability to identify the characteristics of sampling population, they are often not enough to solve problems with highly overlapping interactions between variables. One way to describe multivariate interactions in the multivariate EDA is by using the concept of Bayesian network. A Bayesian network is an acyclic directed graph with one node per variable, where an edge between nodes represents a conditional dependency (Hauschild and Pelikan, 2011). A Bayesian network with n nodes encodes a joint probability distribution of n random variables $X_1, X_2, \ldots, X_n$ (5.4):

$$p(X_1, X_2, \ldots, X_n) = \prod_{i=1}^{n} p(X_i \mid \theta_i) \tag{5.4}$$

where θ_i is the set of variables from which there exists an edge into X_i, and $p(X_i \mid \theta_i)$ is the conditional probability of X_i given θ_i. New candidate solutions are generated by sampling the probability distribution.

Another way to encode multivariate interactions is via Markov networks. The difference between Markov networks and Bayesian networks is the use of undirected connections between variables for Markov networks. A Markov network may sometimes be considerably less complex than a Bayesian network, at least with respect to the number of edges (Mühlenbein, 2008); however, sampling in Markov networks is more difficult than in Bayesian networks. Following these two approaches, the Bayesian optimization algorithm (BOA) (Pelikan, et al., 2000) and the Markovianity-based

optimization algorithm (MOA) (Shakya & Santana, 2008) are developed for multivariate EDAs.

If the interactions between the variables in the optimization problem are not significant, univariate and bivariate EDAs will give better results; however if higher order interactions between the variables emerge, multivariate EDAs should be used to improve the solutions. However, it should be taken into consideration that using more expressive models implies that the solution process will be more computationally expensive.

The 0-1 Multidimensional Knapsack Problem can be modeled by using the EDA variants presented until now. In the 0-1 MKP, the order, or permutation, is not important; only the item selection decision is important for the solution. Therefore, these algorithms are not directly applicable to problems where candidate solutions are represented by permutations such as the quadratic assignment problems, traveling salesman problems and other scheduling problems. These types of problems often contain two specific types of features or constraints: the absolute position of a symbol in a string and the relative ordering of specific symbols (Hauschild & Pelikan, 2011). To deal with this type of problems researches developed EDA-based algorithms, e.g. random key encoding (Bean, 1994), the dependency-tree EDA (dtEDA) (Pelikan, et al., 2007) and the edge histogram based sampling algorithm (EHBSA) (Tsutsui, 2002).

There are some features that distinguish EDA as a stochastic optimization algorithm from other metaheuristics. According to Hauschild and Pelikan (2011), one of the biggest advantages of EDAs over other metaheuristics is their ability to adapt their operators to the structure of the problem, instead of using fixed operators to explore the

space. EDAs also provide information about problem structure, i.e. promising parts of search space, dependency relationships between decision variables, or other important properties of the problem landscape, by means of utilizing probabilistic models. On the other hand, developing explicit probabilistic models in EDA is often more time consuming than using operators in implicit models, such as tournament selection and crossover in EAs. It is also difficult to learn an adequate probabilistic model for the problem; and it may cause creating ineffective algorithms to search the problem space.

5.3 A Representative Example of 0-1 MKP

Suppose there are three knapsacks with upper weight limits of 82, 65, and 51, respectively. A decision maker has to select a set of items from 8 items with different profits and different weights such that the total profit is maximized without exceeding the upper weight limit of each knapsack. Data for the 0-1 MKP example is summarized in Table 7.

The 0-1 MKP can be coded as a general linear 0-1 integer programming problem with nonnegative coefficients, as in equations (5.5 – 5.9).

$$\text{Maximize} \quad 9x_1 + 5x_2 + 19x_3 + 10x_4 + 17x_5 + 11x_6 + 16x_7 + 6x_8 \tag{5.5}$$

$$\text{Subject to} \quad 19x_1 + 14x_2 + 13x_3 + 9x_4 + 15x_5 + 27x_6 + 25x_7 + 18x_8 \leq 82 \tag{5.6}$$

$$20x_1 + 13x_2 + 6x_3 + 10x_4 + 4x_5 + 18x_6 + 27x_7 + 5x_8 \leq 65 \tag{5.7}$$

$$3x_1 + 2x_2 + 5x_3 + 11x_4 + 14x_5 + 23x_6 + 6x_7 + 13x_8 \leq 51 \tag{5.8}$$

$$x_i \in \{0,1\}, i = 1, \ldots, 8 \tag{5.9}$$

Table 7. 0-1 MKP Example

Item	Profit	Constraints		
		1	2	3
1	9	19	20	3
2	5	14	13	2
3	19	13	6	5
4	10	9	10	11
5	17	15	4	14
6	11	27	18	23
7	16	25	27	6
8	6	18	5	13
Upper Weight Limits:		82	65	51

After this example is solved optimally, items 3, 4, 5, 7 and 8 will be selected with an optimum profit of 68.

5.4 Meta-RaPS Dynamic Greedy Rule (DGR) Solution for 0-1 MKP

In this section, the 0-1 MSP example will be solved first by using Meta-RaPS before incorporating a memory mechanism. Meta-RaPS is a two-phase metaheuristic: a constructive phase to create feasible solutions and an improvement phase to improve them. In solving the MKP example with Meta-RaPS, the Dynamic Greedy Rule (DGR) will be used as a priority rule in determining the priorities or order of the items between them (Moraga, et.al, 2005). In this rule, a penalty factor for each item is calculated according to equation (5.10):

$$w_i = \sum_{j=1}^{m} \frac{a_{ij}}{b_j - CW_j}, \text{ for } i = 1, \ldots, n. \tag{5.10}$$

where a_{ij} is the coefficients of item *i* in constraint *j*, b_j is the amount of resource for each constraint *j*, and CW_j is the amount of resource *j* consumed by the items so far; i.e., in the partial solutions. To determine the priority of an item *i*, its profit c_i is divided by its penalty factor, i.e. c_i/w_i. The item with maximum c_i/w_i has the highest priority in the solution process. Because the penalty factors change after each iteration in the construction process, the priorities of the items are updated after each item is added to the partial solution. For example, in the beginning of the process, the priority of item 3 is obtained after the calculations given in equations (5.11-5.12):

$$w_3 = \sum_{j=1}^{3} \frac{a_{3j}}{b_j - CW_j} = \frac{a_{31}}{b_1 - 0} + \frac{a_{32}}{b_2 - 0} + \frac{a_{33}}{b_3 - 0} = \frac{13}{82 - 0} + \frac{6}{65 - 0} + \frac{5}{51 - 0} = 0.35. \tag{5.11}$$

$$\text{priority}_3 = \frac{c_3}{w_3} = \frac{19}{0.34} = 54.5. \tag{5.12}$$

Since in the construction phase of the Meta-RaPS the items are added to the partial solutions, and their order is not important (5.13), the initial priority matrix in Table 8 is created by adding the priority of item *i* to the priority of item *j* if item *i* is selected after *j* was included in the (partial) solution (5.14);.

$$\text{priority}_{ij} = \text{priority}_{ji} \tag{5.13}$$

$$\text{priority}_{ij} = \text{priority}_i + \text{priority}_j \tag{5.14}$$

Meta-RaPS does not select every time the item with the best priority value. The algorithm may accept one with good (not necessarily the best) priority value based on a randomized approach. The priority percentage (p%) is employed to decide the percentage of time the item with the best priority value will be added to the current partial solution,

and (1-p)% of the time an item with a good priority value is randomly selected from a candidate list (CL) that contains items with "good" priorities.

Table 8. The Initial Priority Matrix

Item	1	2	3	4	5	6	7	8
1	-	27.2	69.5	35.9	47.8	25.4	34.1	25.9
2	27.2	-	66.7	33.1	45.0	22.6	31.3	23.1
3	69.5	66.7	-	75.3	87.2	64.9	73.6	65.3
4	35.9	33.1	75.3	-	53.6	31.3	40.0	31.7
5	47.8	45.0	87.2	53.6	-	43.2	51.9	43.6
6	25.4	22.6	64.9	31.3	43.2	-	29.5	21.3
7	34.1	31.3	73.6	40.0	51.9	29.5	-	30.0
8	25.9	23.1	65.3	31.7	43.6	21.3	30.0	-

Table 9. The Meta-RaPS Parameters for the 0-1 MKP Example

Parameter	Value
Priority percentage (p)	0.6
Restriction percentage (r)	0.2
Improvement percentage (i)	0.7
Number of iterations (I)	10

The parameters used in the Meta-RaPS are as given in Table 9. The CL is created for maximization problems by including the ones whose priority values are higher than the lower limit found by equation (5.15):

Lower Limit = Minimum Priority +

(Maximum Priority - Minimum Priority) · (r%) . (5.15)

Checking the feasibility of the (partial) solution in each step of every iteration is very important. That is, the items with the highest priorities and those in the CL must ensure that the (partial) solutions are feasible (i.e., within the limits of the constraints) if added to the (partial) solution.

Meta-RaPS starts by selecting an item randomly as the first item in the partial solution. Because the selected item consumes some of the resources, the priorities in the priority matrix should be updated after each item is added to the partial solution. If, for example, item 5 is selected in the beginning, the updated priorities according to equations (6.10 - 6.13) would be as in Table 10a.

Maximum and minimum priorities of row 5 in Table 6a are 69.9 and 33.8, respectively. If the random number created is smaller than or equal to p%, the item with maximum priority is chosen; otherwise, another item is selected randomly from the CL. Since lower limit [= 33.8 + (69.9 – 33.8) · (0.2)], calculated by using equation (5.14), is equal to 41.02, CL is created by accepting items 4 and 7 whose priorities are larger than lower limit, 42.2 and 41.8 respectively. In the 1st step of iteration 1, because the random number happened to be 0.76 which is greater than p = 0.60, an item from the CL is selected randomly which is for now item 7 as shown in Table 10b.

Table 10a. The Updated Priorities after Selecting Item 5

Item	1	2	3	4	5	6	7	8
1	-	23.5	57.4	29.8	**38.5**	21.3	29.4	21.5
2	23.5	-	54.9	27.3	36.0	18.8	26.9	19.0
3	57.4	54.9	-	61.2	69.9	52.8	60.8	53.0
4	29.8	27.3	61.2	-	42.2	25.1	33.2	25.3
5	38.5	36.0	**69.9**	42.2	-	**33.8**	41.8	34.0
6	21.3	18.8	52.8	25.1	33.8	-	24.7	16.9
7	29.4	26.9	60.8	33.2	41.8	24.7	-	24.9
8	21.5	19.0	53.0	25.3	34.0	16.9	24.9	-

Table 10b. The 1st Step in Iteration 1 of Meta-RaPS

Item	Max Priority	Min Priority	Lower Limit	Max Item	Candidate List	Random Number	p	Decision	Profit
5	69.9	33.8	41.0	3	4, 7	0.76	> 0.60	Select 7	17

After item 7 is added to the partial solution, the priority matrix is again updated, and the column and row of item 5 are deleted. The updated priority matrix for this step is given in Tables 11a and b.

Table 11a. The Updated Priorities after Selecting Item 7

Item	1	2	3	4	5	6	7	8
1	-	14.3	37.3	19.5		13.7	18.0	13.9
2	14.3	-	35.8	18.0		12.2	16.5	12.4
3	37.3	35.8	-	40.9		35.1	39.5	35.4
4	19.5	18.0	40.9	-		17.3	21.7	17.6
5								
6	13.7	12.2	35.1	17.3		-	15.9	11.8
7	18.0	16.5	39.5	21.7		15.9	-	16.1
8	13.9	12.4	35.4	17.6		11.8	16.1	-

Table 11b. The 2nd Step in Iteration 1 of Meta-RaPS

Item	Max Priority	Min Priority	Lower Limit	Max Item	Candidate List	Random Number	p	Decision	Profit
5	69.9	33.8	41.0	3	4, 7	0.76	> 0.60	Select 7	17
7	39.5	15.9	20.6	3	4	0.28	≤ 0.60	Select 3	16

This process is followed until there are no items left without affecting the feasibility of the partial solution. After adding item 3 to the partial solution, it can be seen from the report in Table 12 that item 4 has the highest priority, and there are no items in the CL. However, accepting item 4 makes the partial solution infeasible, and therefore cannot be selected. Because the other items (2, 6, and 8) give the same result, the first

iteration of the algorithm stops. The constructed solution in the first iteration is (5, 7, 3, and 1) and the total profit is 61. The construction phase of Meta-RaPS continues in this fashion until the number of iterations or any other stopping criterion is met.

Table 12. Report for the Construction Phase in Iteration 1 of Meta-RaPS DGR

Item	Max Priority	Min Priority	Lower Limit	Max Item	Candidate List	Random Number	p	Decision	Profit
5	69.9	33.8	41.0	3	4, 7	0.76	> 0.60	Select 7	17
7	39.5	15.9	20.6	3	4	0.28	≤ 0.60	Select 3	16
3	31.4	26.7	27.6	4	1	0.83	> 0.60	Select 1	19
1	5.79	3.59	5.13	4	-	-	-	Stop	9
								Total:	61

The improvement phase of Meta-RaPS is performed only if the feasible solutions generated in the construction phase are within i% of the best unimproved solution value from the preceding iterations (Moraga, et al., 2006). To decide whether to perform the improvement phase after the construction phase for maximization problems or not, the value of Δ in equation (5.16) is calculated;

$$\Delta = WCS + (BCS - WCS) \cdot (i\%) \tag{5.16}$$

where WCS and BCS stand for Worst Constructed Solution and Best Constructed Solution, respectively (Moraga, 2009). If the current solution (CS) is smaller than or equal to the Δ-value, the improvement phase will be executed. At the end of the construction phase for iteration 4, the data collected in this process is summarized in the

Table 13 which shows that an improvement phase is required for iterations 2 and 3. As an example for the improvement phase iteration 3 is selected where the minimum solution value is obtained in the construction phase.

Table 13. Decision for Improvement Phase in Iteration 1 of Meta-RaPS DGR

Iteration	**CS**	**BCS**	**WCS**	**Δ**	**CS vs. Δ**	**Decision**
1	61					
2	60	61	60	60.7	$CS \leq \Delta$	Improve
3	56	61	56	59.5	$CS \leq \Delta$	Improve
4	61	61	56	59.5	$CS > \Delta$	Not Improve

In the improvement phase two different algorithms will be employed: 2-opt and insertion algorithms. In the 2-opt algorithm, the item in the solution is replaced with an item that is not in the solution in a systematic way. To follow this process, the solution is first coded in a binary string, i.e. the solution (5, 7, 3, 1) is coded as (1 0 1 0 1 0 1 0), and the 1's are replaced with 0's. As it can be seen in Figure 2, the better solution is reached by applying the 2-opt algorithm to CS. The improved solution (IS) is generated from (5, 7, 3, 1) with the objective function value of 61 to (5, 7, 3, 6) with objective function value of 63 by replacing items 1 and 6 ($1 \leftrightarrow 6$).

Item	1	2	3	4	5	6	7	8	f(x)	F/NF
CS	1	0	1	0	1	0	1	0	61	
IS	0	0	1	0	1	1	1	0	63	F

Figure 2. Replacing Items in 2-Opt Algorithm

In the other improvement algorithm, insertion, the selected item is inserted to the right or to the left of another item in the solution and items between the old and new places of inserted item are shifted towards the old place of the inserted item in the same order (Arroyo et al., 2008). Other items remain in their positions. In Figure 3, item 7 is inserted to the left of item 4, and items 4 - 6 are shifted towards the old place of item 7. Items 1, 2, 3 and 8 keep their positions.

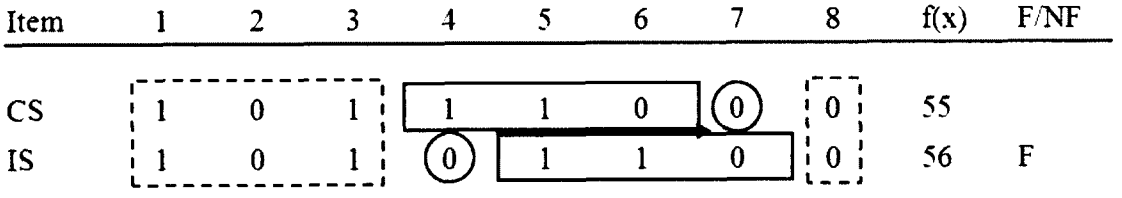

Figure 3. Insertion Items to the Left

Although the Meta-RaPS algorithm does not require the improvement phase for iteration 4 in Table 11, it was carried out for investigation purposes. The effect of the improvement phase can be observed as percentages of increase in the solution objective function values. The optimum value is reached only after applying both algorithms in the improvement phase. Table 14 summarizes the solution report of the 0-1 MKP example by Meta-RaPS for which it could find the optimum value at the 4th iteration.

Table 14. The Meta-RaPS DGR Solution Report of the 0-1 MKP Example

Iteration	Construction Phase	Improvement Phase			
		2-opt	Increase%	Insertion	Increase%
1	61				
2	60	67	11.7	-	-
3	56	61	8.9	-	-
4	61	63	3.3	68*	7.4

5.5 Meta-RaPS EDA Solution for 0-1 MKP

EDA implements a probabilistic learning model as a memory mechanism where it estimates the probability distribution for each decision variable to generate new candidate solutions that replace the old population according to some criteria. This process iterates until termination criteria are met. To be able to create the distribution of the solutions for Meta-RaPS EDA algorithm for the 0-1 MKP example, first a memory set of five feasible solutions (S1 – S5) in Table 15 is generated randomly.

The probability of an item being selected in solutions for this set, P(i), is calculated as in equation (5.17).

$$\text{P(item i)} = \frac{\text{\#item i in solutions}}{\text{\#solutions in memory set}} \tag{5.17}$$

For example, if item 1 is found four times in five solutions then P(item 1) = 4 / 5 = 0.8.

Table 15. The Random Solution Set and Related Information

Item	1	2	3	4	5	6	7	8	f(x)	f(x) Ratio
S1	**0**	0	1	1	0	1	1	0	56	0.21
S2	**1**	1	1	1	1	0	0	0	60	0.23
S3	**1**	1	1	1	0	0	0	1	49	0.19
S4	**1**	1	0	1	1	0	0	1	47	0.18
S5	**1**	1	0	0	1	0	1	0	47	0.18
P(i)	**0.8**	0.8	0.6	0.8	0.6	0.2	0.4	0.4	$\sum$ 259	1.00
wP(i)	**0.156**	0.156	0.127	0.164	0.119	0.043	0.080	0.074		

To include the effect of their objective function values into the process as "weights", the ratio of the objective function value to the total objective function value of solutions in the memory set is calculated for each solution. For example, the objective function value of S1 is 56 and equal to 21% of the total objective function value for all solutions in the memory set which is 259. The contribution of each item to the solution process can be found by taking the mean of ratios of the objective function values for the solutions in which the item is selected. Item 1 is found in solutions 2, 3, 4 and 5, and their ratios are 0.23, 0.19, 0.18 and 0.18, respectively. The contribution of item 1 is the mean of these ratios, which is 0.195. If this contribution is multiplied by P(i), the probability of being selected for item 1, then the (weighted) P(i) is obtained, as $w_{item\ 1}P(\text{item 1}) = 0.8 \cdot 0.195 = 0.156$.

Next step is obtaining the level of interactions between items, e.g. conditional probabilities. The conditional probability, P(item i | item j), which is the probability of

selecting item i given that item j has already been selected in the solution set, is computed for each item by using equation (5.18).

$$P(\text{item i} \mid \text{item j}) = \frac{P(\text{item i} \cap \text{item j})}{P(\text{item j})}. \tag{5.18}$$

For example, assuming item 1 is already selected, the probability of selecting item 3 as the next item for the partial solution is calculated as in (5.19):

$$P(\text{item 3} \mid \text{item 1}) = \frac{\#\text{times both item 3 and item 1 selected (in S2 and S3)}}{\#\text{times item 1 selected (in S2, S3, S4, S5)}} = \frac{2}{4} = 0.5 \tag{5.19}$$

After obtaining the conditional probabilities for all pairs of items, the conditional probability matrix in Table 16 is formed.

Table 16. The Conditional Probability Matrix

Item	1	2	3	4	5	6	7	8
1	-	1.00	0.50	0.75	0.75	0.00	0.25	0.50
2	1.00	-	0.50	0.75	0.75	0.00	0.25	0.50
3	0.67	0.67	-	1.00	0.33	0.33	0.33	0.33
4	0.75	0.75	1.00	-	0.50	0.25	0.25	0.50
5	1.00	1.00	0.33	0.67	-	0.00	0.33	0.33
6	0.00	0.00	1.00	1.00	0.00	-	1.00	0.00
7	0.50	0.50	0.50	0.50	0.50	0.50	-	0.00
8	1.00	1.00	0.50	1.00	0.50	0.00	0.00	-

To transform these two probabilities into an estimation of distribution for items in the memory set, the probability of selecting item *i* given that item *j* has been already

selected is multiplied by the probability of selecting item *i*, i.e. P(item i | item j) · wP(item i). For example, to find 0.064 in Table 17, meaning that is the information within the estimation of distribution for item 3 after item 1 is selected, the probability of selecting item 3 given that item 1 has been selected (= 0.50) is multiplied by the probability of selecting item 3 (= 0.127).

Table 17. The Probabilistic Priority Matrix

Item	1	2	3	4	5	6	7	8
1	-	0.157	**0.064**	0.123	0.089	0.000	0.020	0.037
2	0.157	-	0.064	0.123	0.089	0.000	0.020	0.037
3	0.105	0.105	-	0.164	0.039	0.014	0.026	0.024
4	0.118	0.118	0.127	-	0.059	0.011	0.020	0.037
5	0.157	0.157	0.042	0.110	-	0.000	0.026	0.024
6	0.000	0.000	0.127	0.164	0.000	-	0.080	0.000
7	0.078	0.078	0.064	0.082	0.059	0.022	-	0.000
8	0.157	0.157	0.064	0.164	0.059	0.000	0.000	-

The probabilities in Table 14 constitute the probabilistic priority matrix that will serve as the priority matrix in Meta-RaPS EDA, similar to the DGR values for Meta-RaPS DGR. Progressing in the same fashion with Meta-RaPS DGR and using the same parameters used in Meta-RaPS, the solution (5, 3, 4, 8, and 1) with the total profit of 61 is

obtained at the end of the construction phase in iteration 1 of Meta-RaPS EDA. The detailed report for the last step in iteration 1 is given in Table 18.

As in Meta-RaPS DGR, the current solutions are improved whenever CS is smaller than or equal to the Δ-value calculated using equation (5.15) as shown in Table 19.

Table 18. Report for the Construction Phase in Iteration 1 of Meta-RaPS EDA

Item	Max Priority	Min Priority	Lower Limit	Max Item	Candidate List	Random Number	p	Decision	Profit
5	0.157	0.000	0.031	1,2	3	0.76	> 0.60	Select 3	17
3	0.164	0.014	0.044	4	1,2	0.28	≤ 0.60	Select 4	19
4	0.118	0.011	0.032	1,2	8	0.83	> 0.60	Select 8	10
8	0.157	0.000	0.031	1,2	-	0.58	≤ 0.60	Select 1	6
1	All NF								9
								Total	61

Table 19. Decision Phase for Improvement in Iteration 1 of Meta-RaPS EDA

Iteration	**CS**	**BCS**	**WCS**	**Δ**	**CS vs. Δ**	**Decision**
1	61					
2	56	61	56	59.5	CS ≤ Δ	Improve
3	56	61	56	59.5	CS ≤ Δ	Improve
4	56	61	56	59.5	CS ≤ Δ	Improve

The improvement algorithms are applied to the CSs when the decision is "improve". After using the two improvement algorithms demonstrated earlier in Meta-RaPS DGR, Meta-RaPS EDA algorithm could find the optimum value for the 0-1 MKP example in three iterations. Table 20 summarizes the solution report at the end of 4 iterations.

Table 20. Meta-RaPS EDA Solution Report of the 0-1 MKP Example

Iteration	Construction Phase	Improvement Phase			
		2-opt	Increase%	Insertion	Increase%
1	61	68*	11.5	-	
2	56	68*	21.4	-	
3	56	58	3.6	68*	17.2
4	56	61	8.9	-	-

After the improvement phase at the end of each iteration of the algorithm, the Meta-RaPS EDA memory matrix is updated by replacing the solution found in the current iteration with the solution in the memory matrix. In other words, the memory set will be updated via information obtained after the iterations are completed. The updated probabilistic priority matrix after iteration 1 is shown in Table 21. To memorize and learn the problem structure, Meta-RaPS will need more iterations to converge the probabilistic priority matrix. Only then the Meta-RaPS EDA can have accurate probabilistic priorities to select items in the solution process. Table 22 presents the updated probabilistic priority matrix after 10,000 iterations that helps the algorithm find the optimum solution.

Table 21. The Updated Probabilistic Priority Matrix after Iteration 1

Item	1	2	3	4	5	6	7	8
1	-	0.111	0.112	0.112	0.084	0.000	0.040	0.028
2	0.111	-	0.112	0.112	0.084	0.000	0.040	0.028
3	0.056	0.056	-	0.166	0.063	0.010	0.061	0.042
4	0.056	0.056	0.166	-	0.063	0.010	0.061	0.042
5	0.037	0.075	0.112	0.112	-	0.000	0.040	0.028
6	0.000	0.000	0.166	0.166	0.000	-	0.122	0.000
7	0.037	0.037	0.055	0.112	0.084	0.013	-	0.028
8	0.056	0.056	0.166	0.166	0.063	0.000	0.061	-

Table 22. The Updated Probabilistic Priority Matrix after Iteration 10,000

Item	1	2	3	4	5	6	7	8
1	-	0.007	0.004	0.014	0.007	0.003	0.006	0.009
2	0.005	-	0.007	0.014	0.009	0.003	0.006	0.007
3	0.004	0.010	-	0.013	0.007	0.003	0.007	0.008
4	0.007	0.009	0.006	-	0.008	0.002	0.007	0.008
5	0.006	0.010	0.005	0.014	-	0.002	0.008	0.008
6	0.006	0.010	0.006	0.010	0.005	-	0.003	0.004
7	0.005	0.007	0.006	0.013	0.009	0.001	-	0.009
8	0.008	0.007	0.006	0.014	0.008	0.001	0.008	-

5.6 Comparison of Meta-RaPS with EDA and DGR for 0-1 MKP Example

Because of the memoryless nature of Meta-RaPS DGR, it begins every iteration from the same point, and has no information about the search history. However, in the case of Meta-RaPS EDA, the probabilistic priority matrix serves as a memory and learning mechanism that is updated at every iteration until it converges to its optimum values as iterations proceed. If the items in the probabilistic priority matrix for the 0-1 MKP problem are tracked, it can be easily observed from Figures 4 and 5 that the means of the probabilistic priorities of items in the optimum solution are increasing while other items' means of the probabilistic priorities are decreasing. This observation shows that algorithm memorizes the items with "good" attributes and selects them with higher probabilities, and learns the search space by upgrading the memory matrix after each iteration.

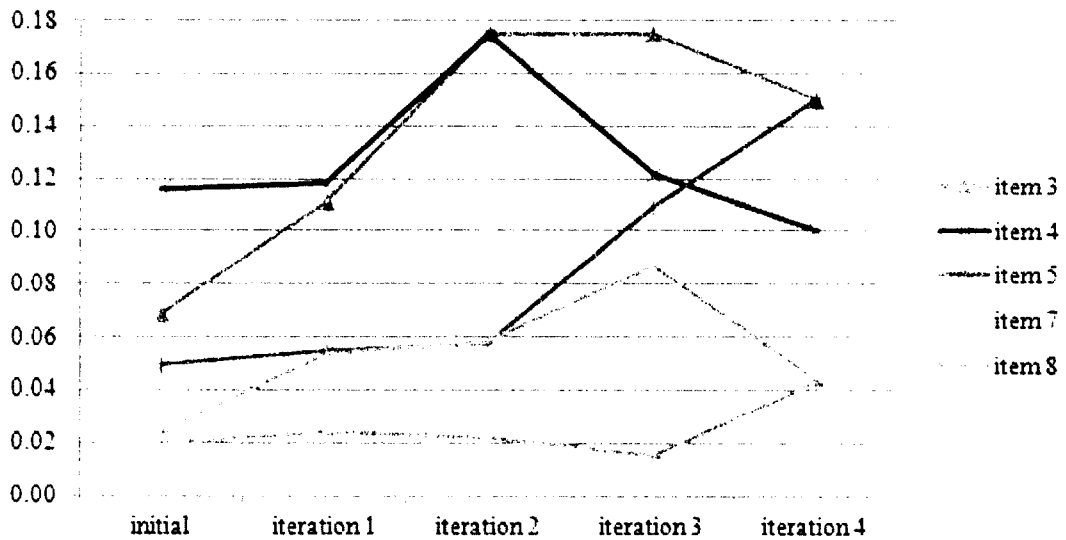

Figure 4. Trend of Probabilistic Priorities of Items Selected in the Optimal Solution

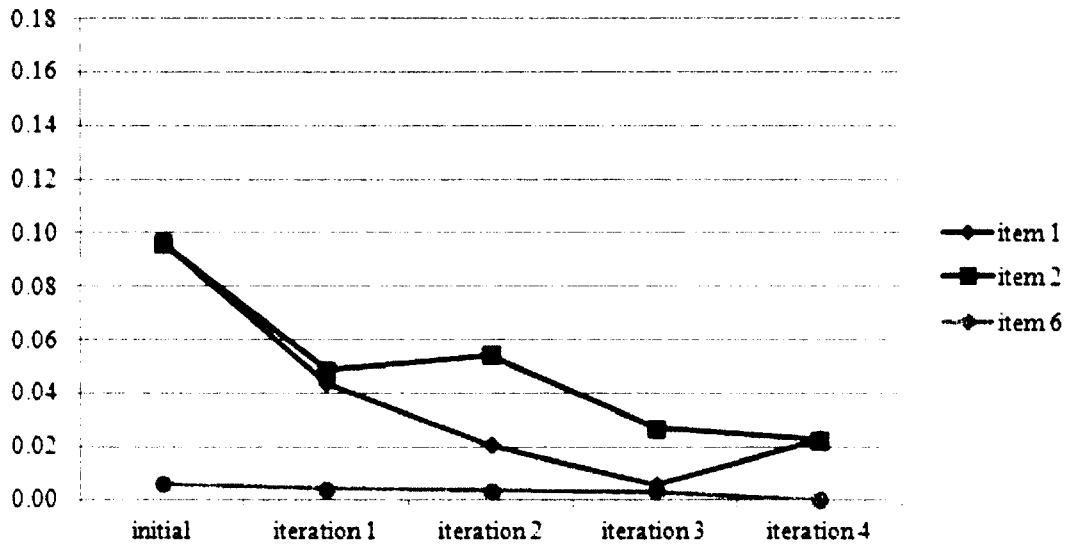

Figure 5. Trend of Probabilistic Priorities of Items Not Selected in the Optimal Solution

Meta-RaPS highly depends on the parameters that are used in the search process. The results obtained in both algorithms could be very different for other choice of parameters. Due to this concern, the same parameter values and also the same random numbers were used for both Meta-RaPS DGR and Meta-RaPS EDA algorithms to be able to compare the performance of each. As a second precaution, all iterations of both algorithms started with the same items for the same reason.

5.7 Meta-RaPS EDA Algorithm

The previous small example presents the role of memory and learning in improving the efficiency of the search process in Meta-RaPS EDA. Because of the probabilistic nature of Meta-RaPS EDA algorithm, the trend for convergence and accuracy of the probabilistic priority matrix is expected to increase with the size of the

instances. The pseudocode of Meta-RaPS EDA in Figure 6 was developed based on the pseudocode of Meta-RaPS in Figure 1.

The core concept of Meta-RaPS EDA is creating probabilities that will serve as priorities in assigning each item to the (partial) solution. The memory matrix is formed by obtaining feasible solutions, and the quality of this matrix is significant in terms of "right" priorities of items. The first step of calculating these priorities is finding the average number of times of each items selected for the solutions in the memory matrix which gives the probability of being in the solution (equation (5.17)).

On the other hand, the goal of solving the 0-1 MKP is to reach the highest profit by selecting appropriate items whose total resource consumptions are under the limit of each knapsack. This fact implies that there should be strong interactions between items since selecting an item affects the selection of other items, which means that the conditional probabilities between items is meaningful as was given in equation (5.18).

Although each of these probabilities carries valuable information, combining these probabilities can have more information for the search process. In addition, including the average value of solutions having item i into the probabilistic model as "weight" will empower the probabilistic priority of each item. Based on these factors, the probabilistic model for Meta-RaPS EDA to solve 0-1 MKP can be shown in (5.20);

```
While (not reached to Memory Matrix size)
    Generate initial solution
    Accept promising solution to Memory Matrix
End While
Build Probabilistic Priority Matrix from Memory Matrix
For iteration ≤ I
    Apply Meta-RaPS rules with priorities from Probabilistic Priority
    Matrix to produce ImprovedSolution
    If ImprovedSolution > BestImprovedSolution then
       Assign ImprovedSolution as BestImprovedSolution
    If ImprovedSolution > WorstSolution in Memory Matrix then
       Replace ImprovedSolution with WorstSolution in Memory Matrix
       Update Probabilistic Priority Matrix
End For
Report BestImprovedSolution
```

Figure 6. Meta-RaPS EDA Pseudo Code

$$P(X_i) = \begin{cases} \frac{1}{n}, \text{ for } i = 1. \\ w_x P(X_i) P(X_i \mid X_j), \text{ for } i > j, j = 1, 2, \ldots, n, i = 2, \ldots, n. \end{cases} \tag{5.20}$$

where i and j are the selection orders, $P(X_i)$ is the probability of item X in the selection order i and w_x is the weight of item X for the memory set. The algorithm starts randomly assigning the first item since the conditional part of probabilities does not take place yet.

With the beginning of selecting the second item, the algorithm employs Meta-RaPS rules with priorities given by the probabilistic model.

In addition to memory, learning is the other important part of a "smart" algorithm. While memory is created from the memory set, learning happens mainly by updating the memory set. Updating activities make it possible for the algorithm to learn the structure of the problem and decide new directions in the search space. There are different criteria to update the memory set, i.e. replacing new solution with a solution in the memory set. The new solution can be replaced with a solution in the memory set selected randomly; or, replacing the solutions can take place only if the objective function value of the new item is greater than the worst objective function value of any solution in the memory set.

Diversity is another way to update the memory set. Diversification in the search space is an important aspect for the solution process in metaheuristics. The level of diversity between solutions can be found by using hamming distance concept. The hamming distance is often used to quantify the extent to which two-bit strings of the same dimension differ (Bookstein, et al. 2002). The diversity between the constructed solution (in Table 11) and the optimum solution for the 0-1 MKP example is calculated as in Table 23. The diversity levels of the new solution and solutions in the memory set can be calculated, and the solution with highest diversity can be selected to replace the existing solutions.

Updating the memory set in Meta-RaPS EDA algorithm is a critical process in integrating learning into the algorithm. Since the small and medium 0-1 MKP instances can be solved quickly, it is important to analyze the memory updating process especially for large size instances.

Table 23. Diversity Calculation between Solutions

Item	1	2	3	4	5	6	7	8
Constructed Solution	1	0	1	0	1	0	1	0
Optimum Solution	0	0	1	1	1	0	0	1
Difference	+	-	-	+	-	-	+	+
Diversity	4							

For this analysis, 1st (01_100x5), 6th (06_100x5) and 22nd (22_100x5) instances from the set of the instances with 100 items and 5 knapsacks are selected randomly and the algorithm is run for 1,000 iterations. To update the memory set after each iteration, the solution in the memory set that will be replaced with the new solution can be selected by four different methods. First, the solution having the minimum value in the memory set can be chosen; secondly, the selection of the solution can be made randomly from the memory set. Third method is to choose the solution with the maximum diversity in the memory set, and last method is applying one of these three methods randomly, named as "All". Figure 7a shows deviations% and Figure 7b shows number of iterations of the three instances after updating memory matrix by each of these methods.

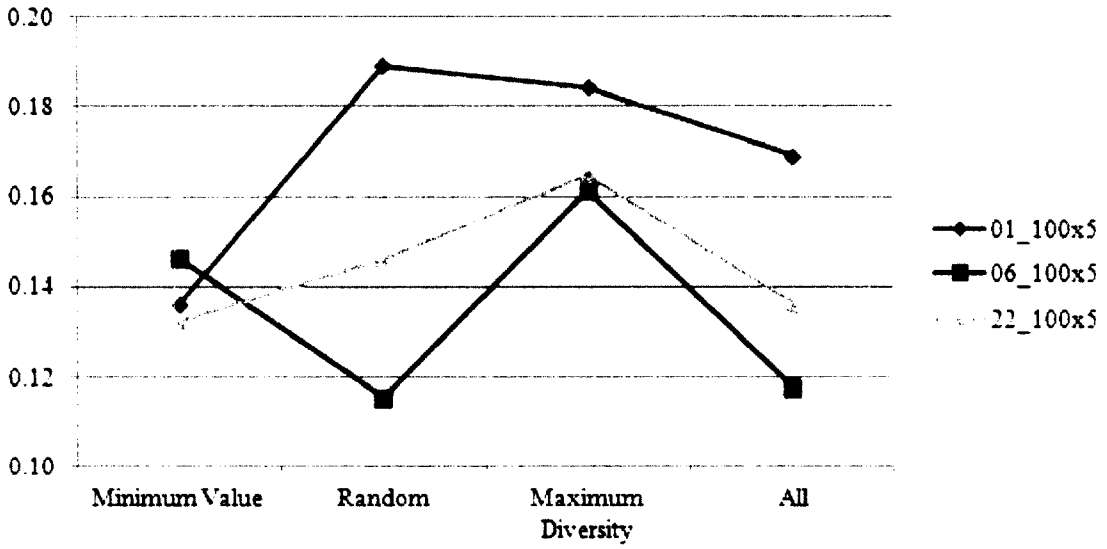

Figure 7a. Deviations% for the Selected Instances after Updating the Memory Matrix by Four Methods

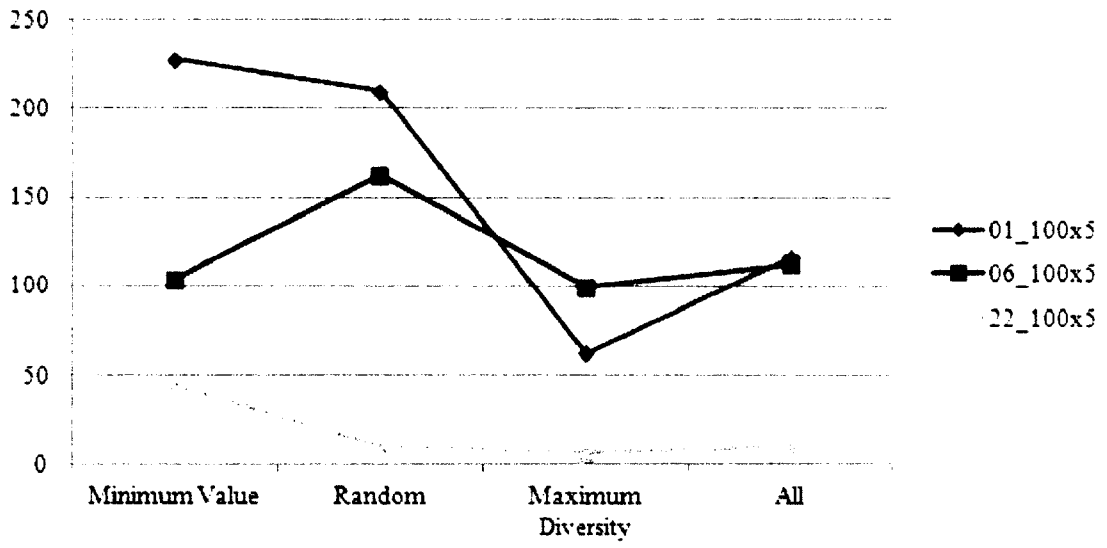

Figure 7b. Number of Iterations for the Selected Instances after Updating Memory Matrix by Four Methods

To update the memory matrix, in terms of deviations%, the first and fourth methods turned out to be better than the other methods, and the third method (selection the solution with the diversity), was found to be the worst method in this analysis. On the other hand, from the figure on number of iterations, it seems that diversity makes the algorithm faster. Fine tuning the update method requires also deeper understanding of the behavior of each method as iterations proceeding. The same instances were solved by changing the updating methods for different number of iterations ranging between 25 and 1000, and the mean deviations% are presented in Figure 8.

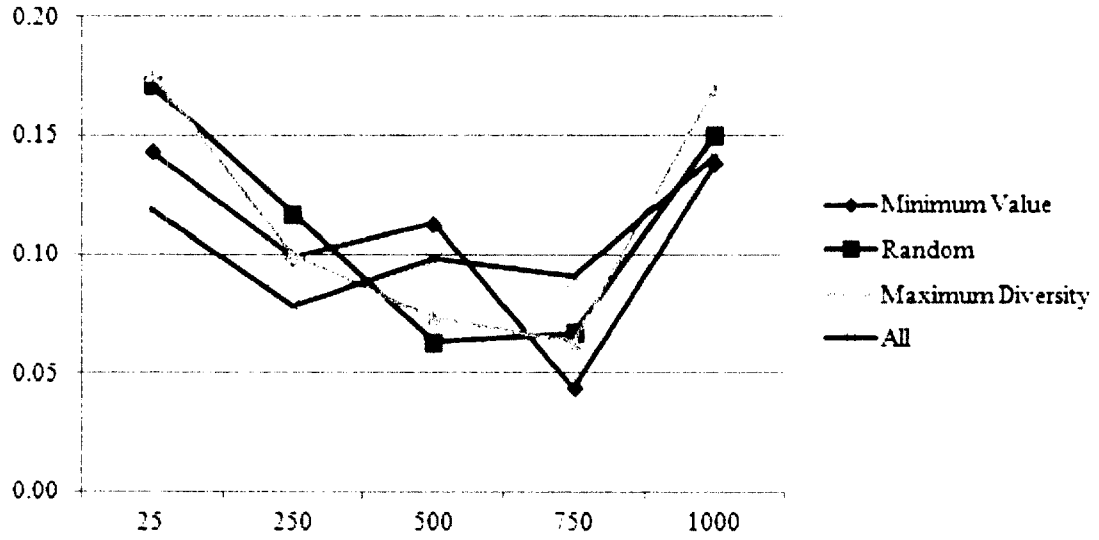

Figure 8. Mean Deviations% after Updating Memory Matrix by Four Methods for Different Number of Iterations

The fourth method, i.e., randomly selecting from the update methods with minimum value, random and maximum diversity, behaves like the regression of the first three methods, and seems to benefit from the advantages of each method in terms of

mean deviations% of the instances. Therefore, in order to exploit all the opportunities created by these three methods, the fourth method (randomly selecting among the three methods) is selected as a mechanism to update the memory matrix.

The last issue in Meta-RaPS EDA is deciding the size of memory set. This is a tradeoff between size of memory set and iteration number which, in most cases, may be interpreted as computational processing time to achieve fast converging and better results versus computational complexity and cost. Finding the best size for the memory set actually adds another parameter, i.e. size of memory set, for the algorithm to deal with. To tune this parameter, 4 instances are solved with different memory set sizes (25, 50, 75, 100 and 125 solutions) and recorded the number of iterations and time used in each process. To be able to compare the required number of iterations in the same scale, their values are divided by the mean values of different memory sizes for the same instance, and the normalized values of the number of iteration are analyzed. To be consistent, the instances with same or close number of items are selected. In Figures 9a and b the normalized values for each instance and their means are respectively shown for memory sizes between 25 and 125 solutions. From these figures, the most effective memory size is determined as 75 in terms of iteration number. After investigating these figures, the size of memory set for the instances with number of items 30 turns out to be 75, which is approximately 2.5 times larger than number of items. The ratio of memory size to number of items can be accepted as a parameter to be tuned before the solution process, and in this case, this parameter is 2.5. It is also observed that, to obtain "good" memory set, i.e. reflecting the structure of the problem, the algorithm should run 4 times the size of memory set, and then solutions in the memory set are selected from these solutions.

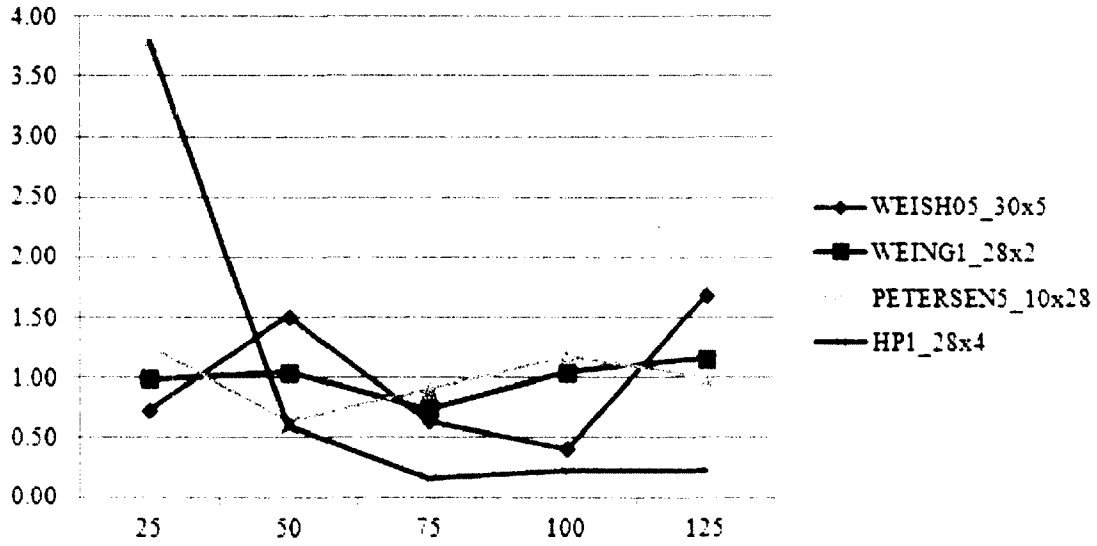

Figure 9a. Normalized Number of Iterations Required for Different Memory Set Sizes

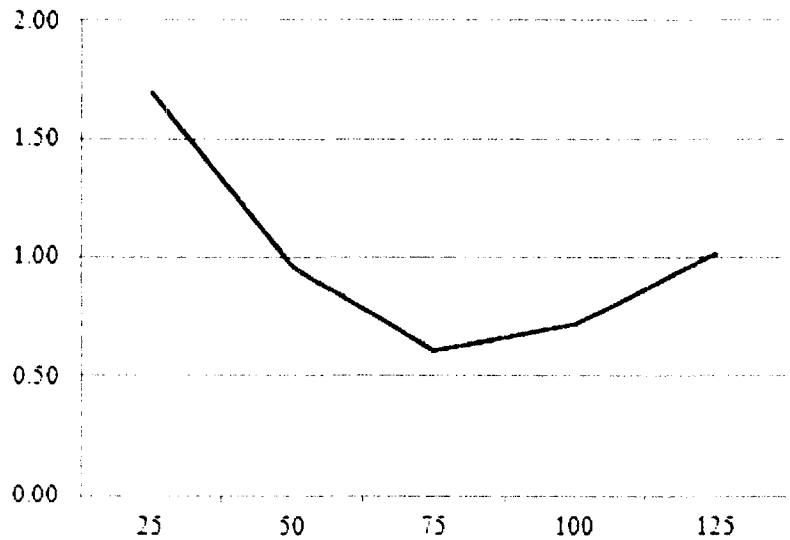

Figure 9b. Trends of Means for Normalized Number of Iterations Required for Different Memory Set Sizes

After completing formalization of the proposed algorithm, Meta-RaPS EDA can be applied to 0-1 MKPs existing in the literature to further evaluate its performance over

the small, medium and large instances to ensure more robust conclusions; however, the same approach can be followed.

5.8 Meta-RaPS EDA for Small and Medium 0-1 MKP Instances

One of the critical aspects of Meta-RaPS EDA is the way the population is created, i.e. the memory set. The solutions in the memory set that form the probability matrix can be generated randomly or by following a greedy rule. Depending on the way the memory matrix is created, Meta-RaPS EDA will have two different versions; Meta-RaPS EDA-R and Meta-RaPS EDA-G.

In the Meta-RaPS EDA-R version, the solutions assigned to the memory matrix are generated totally randomly, and the probabilistic model is produced based on this memory matrix. For the Meta-RaPS EDA-G version, the solutions to form the memory matrix are generated by employing a greedy rule, in this case DGR. With the help of the probabilistic priorities produced from these memory matrices, both versions will be used to solve small and medium 0-1 MKP instances. The parameter settings for small/medium size instances reported in Table 6 was used for both versions of Meta-RaPS EDA, and comparison of their performances is presented in Table 24. *IMean* and *IBest* refer the mean and best values of the Improved Solution (IS), respectively, and *CBest* refers the mean value of the Constructed Solution (CS). Observing the quality of the CSs is important in measuring the effectiveness of the way solutions are created and assigned in the memory matrix.

Meta-RaPS EDA-R could find the optimum values for 54 out of 55 instances previously mentioned in Section 4.3, and WEING7 is the instance that could not be

solved optimally. Its average deviations% from optimum/best solutions found for the CSs and ISs are 1.447% and 0.016%, respectively. The algorithm found the optimum solutions around 9.2 of 10 times on the average of all instances. The average time to solve the small and medium instances was 286.8 seconds on the average of 1199 iterations.

Table 24. Meta-RaPS EDA-R and G Solutions

	Deviation%			Iteration	Time	Optimum	
Version	**IMean**	**IBest**	**CMean**	**Number**	**(Sec.)**	**Frequency**	**Instance**
Meta-RaPS EDA-R	0.016	0.000	1.447	1199	286.83	9.18	54
Meta-RaPS EDA-G	0.001	0.000	0.107	421	120.09	9.84	55
Average	**0.009**	**0.000**	**0.777**	**810**	**203.46**	**9.51**	**54.50**
Std.Dev.	**0.011**	**0.000**	**0.948**	**550**	**117.90**	**0.47**	**0.70**

On the other hand, the average deviation percentages for the Meta-RaPS EDA-G algorithm are 0.107% and 0.001% for the CSs and ISs, respectively. Meta-RaPS EDA-G could find optimum solutions for all small and medium instances, and the mean number of times optimum solutions were found was 9.8 in 10 times for each instance. The average time to solve the instances was 120 seconds on the average of 421 iterations.

When comparing the Meta-RaPS EDA-R and Meta-RaPS EDA-G, it is clear that Meta-RaPS EDA-G produced higher quality solutions in both the IS and CS aspects in lower amount of time than Meta-RaPS EDA-R. It could also find optimum/best solutions for all the small and medium size instances in much shorter time. The standard deviations of the instances' statistics are low for both versions of the Meta-RaPS EDA algorithm, as a sign of being robustness.

These results show that Meta-RaPS EDA-G algorithm is superior to Meta-RaPS EDA-R, in other words, training memory set formed by implementing a greedy rule produces better results than one generated randomly. Therefore, from this point, the Meta-RaPS EDA-G version is accepted as the main version of Meta-RaPS EDA, and will be used to solve large size 0-1 MKP intendances.

5.9 Meta-RaPS EDA for Large 0-1 MKP Instances

With the parameter setting for large instances in Table 6, Meta-RaPS EDA was applied to solve large size 0-1 MKP instances. Detailed solution summary for the first three sets of instances is presented in Table 22. The proposed algorithm could find optimum values for 20, 17 and 10 instances of the 100 items and 5, 10 and 30 knapsacks, respectively, with an average of optimum instances of 15.7 out of 30 instances. The overall average deviations% from optimum/best solution found was 0.142% in an average of 50 minutes and 1872 iterations. The overall average deviations% for CSs was also low, 0.54. Meta-RaPS EDA was also successful at finding optimum/best results in 3.6 of 10 replications as defined in section 4.1. The best average performance of the algorithm, i. e. the best average deviations%, for all instances was 0.084 as shown in Table 25.

Table 25. Meta-RaPS EDA Solution for Large 0-1 MKP Instances

	Deviation%			Iteration	Time	Optimum	
Instance Set	**IMean**	**IBest**	**CMean**	**Number**	**(Min.)**	**Frequency**	**Instance**
100x5	0.045	0.026	0.296	1389	16.19	5.17	20
100x10	0.136	0.078	0.519	1869	33.79	3.90	17
100x30	0.246	0.147	0.804	2357	101.13	1.63	10
Average	**0.142**	**0.084**	**0.540**	**1872**	**50.37**	**3.57**	**15.67**
Std.Dev.	**0.101**	**0.061**	**0.255**	**484**	**44.83**	**1.79**	**5.13**

CHAPTER 6

INCORPORATING Q LEARNING INTO META-RAPS

The second method in adding memory and learning to Meta-RaPS is selected from the machine learning area. Machine learning implements the theory of statistics in building mathematical models of the system by obtaining information from inputs. This is a two-phase process; first is the training phase in which efficient algorithms are used to solve the optimization problem as well as store and process the information derived. In the second phase, new solutions are generated by using the learned model of the problem.

Machine learning approaches can be successfully applied in optimization problems whose output is a sequence of actions, or an optimum policy. Selection of the best actions in intermediate states will not mostly lead to the optimum policy. This type of applications defines the scope of a well known machine learning algorithm known as Q Learning. The Q function is the learned action-value and is defined as the maximum expected, discounted, cumulative reward the decision maker can achieve by following the selected policy. In a Q Learning algorithm, models of the agent or the environment are not required (Monekosso & Remagnino, 2004).

6.1 Literature Review

After Q Learning was introduced by Watkins (1989), many successful applications were presented by researchers in the literature. Cairon and Dorigo (1997) investigated the integration of immediate reinforcements with standard delayed

reinforcements in which reinforcements assigned only when the agent–environment relationship reaches a peculiar state, such as when the agent reaches a target.

The complexity the update process in Q Learning based on lookup tables is bounded by the size of the state-action space. To deal with this issue, Wiering, et al. (1998) created a faster algorithm based on the observation that Q-value updates may be postponed until they are needed. In their Q Learning algorithm, Hirashima, et al. (1999) used an adaptive-sized Q-table based on the Memory Based Learning (MBL). By using the generalization property of the MBL system, the learning effect for a Q-value could be spread to adjacent Q-values, and therefore the number of trial and error actions could be reduced.

The balance between exploration and exploitation is one of the key problems of action selection in Q Learning. Guo, et al. (2004) introduced the Metropolis criterion of Simulated Annealing (SA) algorithm in order to balance exploration and exploitation of Q Learning.

Inspired by the idea that, by using other agents' experiences and knowledge, a learning agent may learn faster, make fewer mistakes, and create some rules for unseen situations, Ahmadabadi and Asadpour (2002) introduced some criteria to measure the expertness of the learning agents, and a new cooperative learning method called weighted strategy sharing (WSS) in which each agent measures the expertness of its teammates and assigns a weight to their knowledge and learns from them accordingly.

Conventional Q Learning techniques are goal dependent; when the reward conditions change, previous learning interferes with the new task that is being learned, resulting in very poor performance. Ollington and Vamplew (2005) presented a new Q

Learning algorithm where the rewards and the environment may change. The algorithm is reward independent, allowing the mechanics of the environment to be learned independently of the task. On the other hand, Fuchida, et al. (2010) proposed a method to propagate negative rewards by taking the absolute value of the next state.

Tesauro and Kephart (2002) investigated a Q Learning utilized by adaptive software agents to make economic decisions such as setting prices in a competitive marketplace. Lee, et al. (2007) employed the Q Learning approach in portfolio management for trading in the stock market. Andrecut and Ali (2001) presented a numerical investigation of the minority game model used to study the competitive interaction of complex adaptive agents in a socioeconomic environment, where the dynamics of the agents is described by the Q Learning algorithm. Zhang and Bhattacharyyaz (2007) produced a Q Learning-based method for supply network agents to search for 'optimal' values of a parameter in their operating policies simultaneously and independently. Jeon, et al. (2011) suggested a routing method for automated guided vehicles in port terminals that uses the Q Learning technique to estimate the waiting times of each vehicle.

There are also studies of the Q Learning algorithm for continuous domains. In the Q Learning algorithm of Millán et al. (2002), the results in robotics domains showed the superiority of the continuous-action Q Learning over the standard discrete-action version in terms of both asymptotic performance and speed of learning. Hagen and Kröse (2003) produced a Neural Q Learning algorithm as a continuous state-action space equivalent of the discrete state-action space Q Learning. Er and Deng (2004) presented a dynamic fuzzy Q Learning (DFQL) method capable of tuning fuzzy inference systems online to

calculate actions and Q-functions to deal with continuous-valued states and actions. A reinforcement distribution method for fuzzy Q Learning to learn a set of fuzzy rules by reinforcement (Bonarini, et al., 2009) and design of fuzzy controllers by ACO incorporated with fuzzy-Q Learning, called ACO-FQ, (Juang & Lu, 2009) are other fuzzy-based Q Learning approaches.

Among the hybrid applications of Q Learning, Monekosso and Remagnino (2004) combined the standard Q Learning technique with a synthetic pheromone introducing a belief factor into the update equation, named the pheromone-Q Learning (Phe-Q) algorithm. Lima, et al. (2007) used the Q Learning algorithm for the constructive phase of GRASP and as generator of the initial population for the GA which was applied to the symmetrical traveling salesman problem. Torre, et al. (2010) offered Q Learning as a mechanism to control how the different evolutionary approaches contribute to the overall search process.

In order to cope with the size of the spaces in Q Learning, various strong approaches to the state and action value function might be needed. Clausen and Wechsler (2000) developed the theory of quad-Q Learning which is applicable to problems that can be solved by "divide and conquer" techniques where the environment was viewed as a hierarchy of states where lower level states are the children of higher level states, and the objective was to maximize the sum of rewards obtained from each of the environments. Castro and Mannor (2010) generalized the classical Q Learning algorithm to an algorithm where the basis of the linear function approximation change dynamically while interacting with the environment. While Bhatnagara and Babu (2008) offered using the two-timescale stochastic approximation methodology in updating Q-values, Langlois and

Sloan (2010) presented a function approximation approach to Reinforcement Learning (RL) via Q function for the Blocks World problem, and they obtained similar learning accuracies with traditional RL, but with better running times. In Hwang et al.'s algorithm (2011), an adaptive resonance method is employed as a cluster to classify input vectors and the results are sent to the Q Learning in order to learn how to implement the optimum actions.

Wang and Silva (2010) presented a Q Learning algorithm with Kalman filtering for decision making in multirobot cooperation where Kalman filter was employed to update Q-values instead of observed rewards. They observed that the algorithm had better performance than the conventional single-agent Q Learning or the team Q Learning in the multirobot domain.

In addition to the applications mentioned before, many other interesting applications were created based on Q Learning, such as channel assignment in mobile communication systems (Nie & Haykin, 1999), multi-agent cooperation for robot soccer (Park, et al., 2001), weightings for optimal control and design problems (Kamali, et al., 2007), robot navigation (Chen, et al., 2008), morphing Unmanned Air Vehicles (Valasek, et al., 2008), adaptive waveform selection in cognitive radar (Wang, et al., 2009), motion control for bionic underwater robots (Lin, et al., 2010) and path selection in disaster response management (Sul, et al., 2011). Learning policies for single machine job dispatching (Wanga & Ushera, 2004), dynamic parallel machine scheduling with mean weighted tardiness (Zhang, et al., 2007) and stochastic resource constrained project scheduling with new project arrivals (Choi, et al., 2007) are some of the Q Learning solution approaches presented in the scheduling area.

A comprehensive tutorial is produced by Kealbling, et al. (1996) and more information about the Q Learning algorithm can be found in Watkins (1989), Watkins and Dayan (1992) and in the textbooks by Mitchell (1997), Sutton and Barto (1998) and Alpaydın (2004).

6.2 Temporal Difference Algorithm – Introduction to Q Learning

Based on available feedback, machine learning can be classified as supervised, unsupervised and reinforcement learning. In supervised learning, correct values are provided by a supervisor; however, in unsupervised learning, there are only input data and no supervisor. The goal is to obtain the regularities in the input which is defined as density estimation in statistics. Clustering is a method for density estimation. In RL, unlike supervised leaning, the machine is not told which actions to take but has to discover which actions yield the most reward (Yao & Liu, 2005). The modern science of RL has emerged from a synthesis of notions from four different fields: classical Dynamic Programming (DP), Artificial Intelligence (AI), stochastic approximation, and function approximation (Gosavi, 2009). What RL algorithm does is evaluating the goodness of policies and learning from the good action sequences to create a policy. Trial-and-error search and delayed reward are the two most important unique characteristics of RL (Yao & Liu, 2005).

The RL process creates a sequence of actions; it indeed applies a Markov decision process (MDP) to model the agent. However, there is a significant difference between these two cases. While in MDP the sequence of signals is produced by an external process, in the RL algorithm the agent itself generates the sequence of actions (Alpaydın,

2004). A more realistic approach would be to explore the environment and use this information for updating the current state. These types of RL algorithms are defined as Temporal Difference (TD) algorithms (Sutton, 1988). In TD we look at the difference between the current estimate of the value of a state and the discounted value of the next state and the reward. TD approach is historically based on animal learning psychology and artificial intelligence (Klopf 1972; Samuel, 1959).

A TD algorithm is a combination of Monte Carlo (MC) and dynamic programming (DP) ideas. TD methods can learn directly from the experience without a model of the environment's dynamics like MC methods, and, like DP, TD can update the estimated values based on other learned values, without waiting for a final outcome. The relationship between TD, DP, and MC methods is the recurring concept of RL (Sutton & Barto, 1998). TD updates the estimates of the value function V^{π} for a given policy π. The simplest TD method, known as TD(0), is presented in equation (6.1);

$$V(s_t) \leftarrow V(s_t) + \alpha\,[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)] \tag{6.1}$$

where r_{t+1} is the actual return at time (t+1), α is step-size, or learning parameter and γ is the discount parameter. Note that the definition of an optimal policy in equation (6.1) is inspired by considering Bellman's equation (6.2), which forms the foundation for many dynamic programming approaches to solving MDPs.

$$(\forall s \in S)V^*(s) = E[r(s,\pi(s)) + \gamma V^*(\delta(s,\pi(s)))] \tag{6.2}$$

Bellman (1957) showed that the optimal policy π^* satisfies the equation (6.2) and that any policy π satisfying this equation is an optimal policy. The main contribution of Bellman's work was to show that the computational burden of an MDP could be dramatically reduced via DP (Gosavi, 2009).

Unlike MC, TD will not wait until the end of the episode to determine $V(s_t)$, instead, it just needs to wait only until the next time step. This feature is the most obvious advantage of TD methods over MC methods. In practice, TD has usually been found to converge faster than constant-α MC methods on stochastic problems (Sutton & Barto, 1998).

Different from DP, TD does not need a model of the environment, reward and next-state probability distributions. If all the rewards and next-state probability distributions are known, DP could be used instead of the RL algorithm. However, obtaining this information can be very costly and seldom possible. According to Alpaydın (2004), the RL has two advantages over classical DP: first, while learning, it can intensify the important parts of the search space and ignore the other parts; and second, it can implement function approximation methods to model the problem and learn faster. Q Learning may be accepted as stochastic approximations to DP (Jaakkola, et al., 1994).

In complex problems with several governing random variables, it is usually difficult to compute the values of the transition probabilities. This phenomenon is called the curse of modeling. In problems with large dimensions, storing or managing these values becomes challenging. This is called the curse of dimensionality. DP breaks down on problems which suffer from any one of these curses because it requires all these values. RL can generate near-optimal solutions by making inroads into problems that suffer from any of these curses and cannot be solved by DP (Gosavi, 2009).

When the feedback used is from one state transition of the MC, the algorithm is named a TD(0) algorithm, as in equation (6.1). When the feedback is from multiple

transitions, the algorithm is then referred to as a TD(λ) algorithm. In TD(λ), we have that feedback = $r_i + \lambda r_{i+1} + \lambda^2 r_{i+2} + \ldots$; where r_i is the immediate reward received in the i^{th} iteration. The Q Learning algorithm is derived from the definition of Q values and TD(0) algorithm (Gosavi, 2009).

6.3 Q Learning

TD focuses on the tansitions from state to state and the learned values of states. If instead, the tansitions from state-action pair to state-action pair are considered, their learned values will bring us to the Q Learning algorithm; one of the most important breakthroughs in RL (Sutton & Barto, 1998). In Q Learning algorithm, the learned action-value function, Q, directly approximates Q*, the optimal action-value function. On the other hand, TD algorithms learn by iteratively reducing the differences between the estimated values produced by the agent at different times. In this sense, Q Learning is a special case of a general class of TD algorithms (Mitchell, 1997).

In the Q Learning algorithm, the value of evaluation function Q(s, a) is defined as the maximum discounted cumulative reward that can be achieved starting from state *s* and applying action *a* as the first action. The Q value is the reward received immediately after selecting action *a* from state *s*, plus the value, discounted by γ, of following the optimal policy. If the agent learns the Q function, it will be able to select the action that maximizes Q(s, a) among available actions in its current state.

In the terminology of the Q Learning algorithm, the decision maker is called "agent". There are several possible "states" for the agent to move from one to another. The "environment" is the current state in which the agent interacts and makes decisions.

The agent has a set of possible, or feasible, actions that affect both the "reward" and the next state. Once an action is taken, the state will be changed. For each action the agent receives feedback, called the "reward". The rewards are delayed, and required for the agent to learn the system. To solve the optimization problem the agent learns the best course of actions that have the maximum cumulative reward. The sequence of actions from the first state to the terminal state is called episode.

$$Q(s_t,a_t) \leftarrow (1-\alpha)\ Q(s_t,a_t) + \alpha\left[r_{t+1} + \gamma \max_{a_{t+1}} Q(s_{t+1},a_{t+1}) \right] \tag{6.3}$$

In the Q Learning transition, equation (6.3), $Q(s_t, a_t)$ is nominated as the cumulative quality or reward of action taken in state *s* for time *t*. r_{t+1} is the reward received when the action a is taken at time (t + 1). $Q(s_{t+1}, a_{t+1})$ is the value for the next state, and has a higher chance of being correct. If s_{t+1} is terminal, then $Q(s_{t+1}, a_{t+1})$ is defined as zero. α is the learning factor, $0 < \alpha < 1$, which is gradually decreased in time to converge. It has been shown that as α is gradually decreased in time for convergence, this algorithm converges to the optimal Q* values (Watkins & Dayan, 1992). More general convergence results were proved later by Jaakkola, et al. (1994) and Tsitsiklis (1994).

The learning factor α is a function of the number of iterations. Let α^i denote the main learning rate in the i^{th} iteration. Some commonly used examples for step-sizes are: $\alpha^i = a / (b + i)$ where for instance a and b are constants and $log(i) = i$ (Gosavi, 2009).

Besides the learning factor, Q values for the next state, i.e. $Q(a_{t+1}, s_{t+1})$, are discounted by a discount factor γ, where $0 \leq \gamma < 1$, since these Q values will happen in the next step, in other words, in the future (Junior et al., 2008). The discount concept

essentially measures the present value of the sum of the rewards earned in the future over an infinite time, where γ is used to discount money's value horizon, in equation (6.4):

$$\gamma = \left(\frac{1}{1+\mu}\right)^{l} \tag{6.4}$$

where μ is the rate of interest. The right part of the equation is raised to the power 1 because in the MDP the time duration of each transition is fixed at 1. If $\gamma = 0$, only the immediate reward is considered. As γ approaches 1, future rewards are given greater emphasis relative to the immediate reward.

After discounting, if we add it to the immediate reward $[r_{t+1} + \gamma \max Q(s_{t+1}, a_{t+1})]$ then this term can be accepted as an estimated value of the action in the next step which is called backup because it can be viewed as "backing it up" to revise the estimate for the value of a current action (Alpaydın, 2004) .

In the process of Q Learning algorithm, all Q values are stored in a lookup table, and initially all $Q(s_t, a_t) = 0$ for all actions. Assuming all Q values are initialized to zero, Q Learning has two general properties that hold for any deterministic MDP (Mitchell, 1997). First, the Q values never decrease during training, and second, throughout the training process every Q value will remain in the interval between zero and its true Q value.

A Q Learning algorithm can be proven to converge to their optimal Q values when the estimated Q values for each state-action pair are represented by a lookup table with a distinct entry for each state-action pair. The key idea of the proof of convergence of Q Learning is that the table entry Q(s, a) with the largest error must have its error reduced by a factor of γ whenever it is updated. The reason is that its new value depends only in part on error-prone Q estimates, with the remainder depending on the error-free

observed immediate reward r. According to convergence theorem of Q Learning for deterministic MDP with bounded rewards, $|r(s, a)| \leq c$ for all s, a, first consider that the Q Learning agent uses the training rule of equation (6.3), initializes the look-up table Q(s, a) to arbitrary finite values, and uses a discount factor γ such that $0 \leq \gamma < 1$, for the n^{th} update. If each state-action pair is visited infinitely often, the estimates of $Q_n(s, a)$ converges to the real values of $Q(s, a)$ as $n \rightarrow \infty$, for all s, a (Mitchell, 1997).

The most constraining assumption in Q Learning is that the Q function is represented as a lookup table with a discrete entry for every state-action pair. However, there are a number of problems with this lookup table approach (Alpaydın, 2004):

- Increasing the numbers of states and actions makes the size of lookup table quite larger.
- Instead of discrete entries for state-action pair, states and actions may be continuous.
- When the search space is large, more episodes may be needed to fill the entries of the lookup table with acceptable accuracy.

To be able to overcome these problems in Q Learning, other practical algorithms are often combined with the Q Learning training rules, such as regression models, function approximation methods, artificial neural networks and clustering. In practice, a number of successful RL systems have been developed by incorporating such algorithms in place of the lookup table (Mitchell, 1997).

Updating $Q(s_t, a_t)$ values in the lookup table can be carried out in two ways; off-policy and on-policy. In off-policy control, the policy being evaluated to update the Q values can change in every iteration. In on-policy control, a unique policy is evaluated for

some time during the learning (Gosavi, 2009). On-policy updating version of the Q Learning algorithm is called SARSA. The term comes from the initial letters of State, Action, Reward, State, and Action (Battiti, et al., 2008). The SARSA algorithm was first introduced by Rummery and Niranjan (1994), who called it modified Q Learning. The name "SARSA" was used first by Sutton (1996).

Q Learning methods are the most widely used RL methods, probably due to their great simplicity. They can benefit from the experience generated from interaction with an environment, and be applied with a minimal amount of computation (Sutton & Barto, 1998). The novel aspect of Q Learning is that it assumes the agent must move about the real world and observe the consequences. The primary concern is usually the number of real-world actions that the agent must perform to converge to an acceptable policy, rather than the number of computational cycles it must expend (Mitchell, 1997).

6.4 Meta-RaPS Q Solution for 0-1 MKP Example

To demonstrate how the Meta-RaPS Q Learning algorithm works, it was applied to the 0-1 MKP example in Section 6.3. In this algorithm, the agent, i.e., decision maker, will select the next item to add to the partial solution as an action in the current state. The decision of agent depends on the current and next states as well as the rewards, i.e. weights or priorities, of the feasible items. After one selection, the agent moves to the next state to take another action (selecting another item), until the current episode, i.e. the constructed solution, is completed. For this example, learning factor α and discount factor γ are assumed to be 0.7 and 0.1, respectively.

To begin the algorithm, first the lookup table for the Q Learning matrix is created and initialized with zeros, i.e. $Q(s_t, a_t) = 0$, for all actions and all states. While progressing, the $Q(s_t, a_t)$ values will be stored in this matrix. The Q Learning matrix after 1000 episodes presented in Table 26 will be used to explain the updating operations of Q values by equation (6.3). Note that in the Q Learning matrix, some cells are filled with zeros since it is not possible to select an item in every state to reach optimal solutions for 0-1 MKPs, i.e. not all items can be accepted in the solution set due to feasibility concerns. However, for permutation problems, such as scheduling problems where the goal is to create optimum ordering of jobs, in each state a job would be selected and therefore the matrix would be filled with numbers other than zeros.

Table 26. The Q Learning Matrix after 1000 Episodes for 0-1 MKP Example

$Q(s_t, a_t)$	1	2	3	4	5	6	7	8
1	39.496	24.242	64.365	35.355	36.908	14.828	25.164	18.336
2	13.030	14.451	44.368	20.676	25.461	17.164	20.275	14.687
3	9.014	6.929	25.214	19.539	15.563	7.454	9.474	5.991
4	9.196	4.197	17.963	5.319	9.240	0.000	4.459	6.665
5	0.000	9.161	0.000	0.000	0.000	0.000	0.000	0.000
6	0.000	7.047	0.000	0.000	0.000	0.000	0.000	0.000
7	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
8	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

The process begins randomly by selecting an item for the first or current state, and another item as the next feasible item for the next action. If in the current state, $t = 1$, the

selected item is 6, the value of Q(1, 6) is calculated by using equation (6.3) based on all feasible actions after item 6 in Figure 10, and the Q Learning matrix is updated by replacing new Q(1, 6) with the previous one, as in the Table 27. In this calculation, r_{t+1} is 19, the weight of item 3 that has the maximum Q value in state t + 1.

$$
\begin{aligned}
Q(1,6) &= (1 - 0.7) \cdot Q(1, 6) + 0.7 \cdot [r_2 + 0.1 \cdot \max\{Q(2, 1), Q(2, 2), Q(2, 3), Q(2, 4), \\
&\qquad Q(2, 5), Q(2, 7), Q(2, 8)\}] \\
&= 0.3 \cdot 14.828 + 0.7 \cdot [19 + 0.1 \cdot \max\{13.030, 14.451, 44.368, 20.676, \\
&\qquad 25.461, 20.275, 14.687)\}] \\
&= 20.860
\end{aligned}
$$

Figure 10. Calculating Q Value for t = 1

Table 27. The Updated Q Learning Matrix after t = 1

$Q(s_t, a_t)$	1	2	3	4	5	6	7	8
1	39.496	24.242	64.365	35.355	36.908	**20.860**	25.164	18.336
2	13.030	14.451	44.368	20.676	25.461	17.164	20.275	14.687
3	9.014	6.929	25.214	19.539	15.563	7.454	9.474	5.991
4	9.196	4.197	17.963	5.319	9.240	0.000	4.459	6.665
5	0.000	9.161	0.000	0.000	0.000	0.000	0.000	0.000
6	0.000	7.047	0.000	0.000	0.000	0.000	0.000	0.000
7	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
8	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

Once item 6 is assigned, the next item is chosen in state 2 among the feasible unassigned items, i.e., we seek the value of Q(2, 3) to update the Q Learning matrix. Again, after following calculations carried out according to equation (6.3) in Figure 11, where r_{t+2} is the weight of the item 4, whose Q value in state t + 2 is maximum, the new value of Q(2, 3) is replaced with the former value in the Q Learning matrix, as in the Table 28.

$$
\begin{aligned}
Q(2,3) &= (1-0.7)\cdot Q(2,3) + 0.7\cdot[r_3 + 0.1\cdot \max\{Q(3,1), Q(3,2), Q(3,4), Q(3,5), \\
&\qquad Q(3,7), Q(3,8)\}] \\
&= 0.3\cdot 44.368 + 0.7\cdot[10 + 0.1\cdot \max\{9.014, 6.929, 19.539, 15.563, \\
&\qquad 9.474, 5.991\}] \\
&= 21.678
\end{aligned}
$$

Figure 11. Calculating Q Value for t = 2

Table 28. The Updated Q Learning Matrix after t = 2

$Q(s_t, a_t)$	1	2	3	4	5	6	7	8
1	39.496	24.242	64.365	35.355	36.908	**20.860**	25.164	18.336
2	13.030	14.451	**21.678**	20.676	25.461	17.164	20.275	14.687
3	9.014	6.929	25.214	19.539	15.563	7.454	9.474	5.991
4	9.196	4.197	17.963	5.319	9.240	0.000	4.459	6.665
5	0.000	9.161	0.000	0.000	0.000	0.000	0.000	0.000
6	0.000	7.047	0.000	0.000	0.000	0.000	0.000	0.000
7	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
8	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

After selecting item 4, in state 3 we update the value of Q(3, 4) by assigning an item so that the solution remains feasible. In this case, only adding one of the items 1, 2 and 7 keeps the solution feasible (Figure 12).

$$
\begin{aligned}
Q(3, 4) &= (1 - 0.7) \cdot Q(2, 3) + 0.7 \cdot [r_4 + 0.1 \cdot \max\{Q(4, 1), Q(4, 2), Q(4, 7)\}] \\
&= 0.3 \cdot 19.539 \quad + 0.7 \cdot [9 + 0.1 \cdot \max\{9.196, 4.197, 4.459\}] \\
&= 12.805
\end{aligned}
$$

Figure 12. Calculating Q Value for t = 3

Item 1 is the last feasible item added for the solution in this episode. Although there will not be any item to select for the next state, we can calculate the value of the last state-action pair, Q(4, 1) in Figure 13:

$$
\begin{aligned}
Q(4, 1) &= (1 - 0.7) \cdot Q(4, 1) + 0.7 \cdot [r_5 + 0.1 \cdot 0] \\
&= 0.3 \cdot 9.196 \\
&= 2.756
\end{aligned}
$$

Figure 13. Calculating Q Value for t = 4

After assigning items 6, 3, 4 and 1 there are no other feasible items to add to the solution, thus the algorithm is stopped for this episode. Table 29 presents the Q Learning matrix after 1,001 episodes.

Table 29. The Q Learning Matrix after 1001 Episodes for 0-1 MKP Example

$Q(s_t, a_t)$	1	2	3	4	5	6	7	8
1	39.496	24.242	64.365	35.355	36.908	**20.860**	25.164	18.336
2	13.030	14.451	**21.678**	20.676	25.461	17.164	20.275	14.687
3	9.014	6.929	25.214	**12.805**	15.563	7.454	9.474	5.991
4	**2.756**	4.197	17.963	5.319	9.240	0.000	4.459	6.665
5	0.000	9.161	0.000	0.000	0.000	0.000	0.000	0.000
6	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
7	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
8	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

In the Meta-RaPS Q Learning algorithm, the converged Q Learning matrix is used as the probabilistic priority matrix for Table 22, created for the Meta-RaPS EDA algorithm in Chapter 5, to determine the priorities of each item in partial solutions, and the process Meta-RaPS followed in assigning new items. After each iteration, the Q Learning matrix is updated by accepting the improved solution as an episode of Q values for selected items and their states.

6.5 Meta-RaPS Q Algorithm

The Q Learning matrix is the memory and learning mechanism that provides the priorities necessary to select the items by Meta-RaPS. Since this mechanism draws its strength from the transition equation (6.3), this equation has a key role in obtaining a matrix which should present the "right" priorities for choosing the order of items. At this point, two issues related to equation (6.3) emerge; selection of the next Q values and definition of rewards.

In the 0-1 MKP example, next item is selected according their Q values, i.e. the item with the maximum Q value. However, by accepting the item with the maximum Q value the algorithm may be stuck in local regions, and cannot explore other promising areas. A precaution to remove this risk may be to select the next item randomly instead of looking for the one with maximum Q value. The preliminary analysis of both approaches showed that in selecting the next item, randomness strategy is superior to elitist strategy. According to this result, the transition equation (6.3) is modified as presented in equation (6.5). The pseudocode summarizing the Meta-RaPS Q process is developed in Figure 14.

$$Q(s_t,a_t) \leftarrow (1-\alpha)\ Q(s_t,a_t) + \alpha\left[r_{t+1} + \gamma \underset{a_{t+1}}{\text{random}}\, Q(s_{t+1},a_{t+1})\right] \tag{6.5}$$

Another issue in the transition equation is how to identify the rewards. In the 0-1 MKP example, rewards were simply accepted as the weights of items. If this approach is used with the modified transition equation (6.5) the algorithm will be totally independent from any other greedy rule. On the other hand, to increase the chance of better results, DGR can also be applied in calculating rewards for this problem. These two approaches form two versions of Meta-RaPS Q algorithm to solve 0-1 MKP; Meta-RaPS Q-W and Meta-RaPS Q-G, respectively and will be discussed next.

While (Q Learning Matrix not converged)

Initialize Q Learning Matrix with zeros

Do for each episode

Calculate Q values for current episode by transition equation:

$$Q(s_t, a_t) \leftarrow (1-\alpha)\,Q(s_t, a_t) + \alpha\left[r_{t+1} + \gamma \underset{a_{t+1}}{\text{random}}\, Q(s_{t+1}, a_{t+1})\right]$$

End While

For iteration $\leq$ I

Apply Meta-RaPS rules with priorities from Q Learning matrix to produce ImprovedSolution

If ImprovedSolution > BestImprovedSolution then

Assign ImprovedSolution as BestImprovedSolution

Update Q Learning Matrix by accepting ImprovedSolution as an episode

End For

Report BestImprovedSolution

Figure 14. Meta-RaPS Q Pseudo Code

6.6 Meta-RaPS Q for Small and Medium 0-1 MKP Instances

As in the case of the Meta-RaPS EDA, the success of the Meta-RaPS Q algorithm depends on the quality of the Q Learning matrix. The key element in determining this quality is the reward in the Q transition equation (6.5). The weights of items, like in the 0-1 MKP example, can be used as the rewards to update the Q values of state-action pairs, or they can be produced by employing a greedy rule. Depending on how the rewards are accepted, Meta-RaPS Q will have two different versions: Meta-RaPS Q-W where weights of items are accepted as rewards and Meta-RaPS Q-G where rewards are generated using DGR. Table 30 summarizes the results of both algorithms for small/medium 0-1 MKP problems.

Meta-RaPS Q-W could find the optimum values for 45 instances of 55 with the average deviation percentages from optimum/best solutions for the CSs and ISs of 1.496% and 0.214%, respectively. The average time and iterations to solve the small and medium instances are 747 and 2074 seconds respectively. The algorithm based on the item weights found the optimum solutions for the instances 6.8 of 10 times on average.

Table 30. Meta-RaPS Q Solutions for Small/Medium 0-1 MKP Problems

	Deviation%			Iteration	Time	Optimum	
Version	**IMean**	**IBest**	**CMean**	**Number**	**(Sec.)**	**Frequency**	**Instance**
Meta-RaPS Q-W	0.214	0.089	1.496	2090	774.05	6.82	45
Meta-RaPS Q-G	0.045	0.003	0.596	2074	209.19	7.96	52
Average	**0.130**	**0.046**	**1.046**	**2082**	**491.62**	**7.39**	**48.50**
Std.Dev.	**0.120**	**0.061**	**0.636**	**11**	**399.42**	**0.81**	**4.95**

Using with DGR, the average deviations of the Meta-RaPS Q-G algorithm are 0.596% and 0.045% for the CSs and ISs, respectively. Meta-RaPS Q-G could reach the optimum solutions of 52 small and medium instances; PETERSEN7, WEING7 and WEISH18 were the instances not solved by the algorithm. The mean number of times optimum solutions found was around 8 in 10 times for each instance. The average time to solve instances was 209 seconds for the average of 2074 iterations.

Comparing the performances of both algorithms, it is obvious that Meta-RaPS Q-G is superior to Meta-RaPS Q-W as it produces higher quality results in all aspects. As in the case of EDA, training memory set by implementing a greedy rule produced better results than using just the weights of items. Due to these facts, the Meta-RaPS Q-G

version is accepted as the main version of Meta-RaPS Q to apply for large size 0-1 MKP intendances.

6.7 Meta-RaPS Q for Large 0-1 MKP Instances

With the parameter settings in Table 6 Meta-RaPS Q was run to solve large size 0-1 MKP instances. In Table 31, the Meta-RaPS Q algorithm obtained the optimum values for 22, 10 and 6 instances of the 100 items and 5, 10 and 30 knapsacks, respectively. The average optimum instance was 12.7 of 30 instances for the first three sets. The overall average deviations from optimum/best solution found so far was 0.452% in an average of 87 minutes and 4176 iterations. The overall average deviation percentage for CSs was 1.17; not as low as in Meta-RaPS EDA. Meta-RaPS Q could reach the optimum/best results in 2.4 of 10 replications. The best average performance of the Meta-RaPS Q, i. e. the best average deviations for all instances was 0.273.

Table 31. Meta-RaPS Q Solution for Large 0-1 MKP Instances

	Deviation%			Iteration	Time	Optimum	
Instance Set	**IMean**	**IBest**	**CMean**	**Number**	**(Min.)**	**Frequency**	**Instance**
100x5	0.143	0.058	0.653	3440	32.79	4.97	22
100x10	0.493	0.281	1.261	4454	65.58	1.50	10
100x30	0.721	0.479	1.599	4634	163.80	0.60	6
Average	**0.452**	**0.273**	**1.171**	**4176**	**87.39**	**2.36**	**12.67**
Std.Dev.	**0.291**	**0.211**	**0.479**	**644**	**68.17**	**2.31**	**8.33**

CHAPTER 7

INCORPORATING PATH RELINKING INTO META-RAPS

In the EDA and Q based memory and learning mechanisms presented in previous chapters, the first step was to create a memory set for the algorithm to learn the problem structure. As the third approach to attempt gaining memory and learning, Path Relinking (PR) will be employed in Meta-RaPS as a post optimization procedure. In this approach, there will be no need for a memory set, and learning will take place only after producing solutions by Meta-RaPS.

The approach is named Path Relinking because it generates a path between solutions linked by series of moves during a search to incorporate attributes of the guiding solution while recording the objective function values (Glover & Laguna, 1997). In the PR algorithm a trajectory, or a path, is created between two solutions, called initial and guide solutions, to create new solutions. While progressing, the initial solution gradually transforms in the guide solution by incorporating the attributes of the guide solution.

7.1 Literature Review

Path relinking was originally proposed by Glover (1996) as a way to explore trajectories between elite solutions obtained by TS or Scatter Search (SS), and later Laguna and Martí (1999) applied PR within GRASP. The PR became an attractive approach applied as an intensification strategy in GRASP (Resende & Ribeiro, 2003), as a post-optimization step (Deng and Bard, 2011; Ribeiro, Uchoa, & Werneck, 2002;

Villegas, et al., 2011), or as both intensification and post-optimization strategies (Resende & Werneck, 2002).

In the literature, GRASP and PR applications are produced by researchers for many optimization problems, such as scheduling (Alvarez-Valdes, et al., 2008, Arroyo, et al., 2008, Bozejko, 2010), max–min diversity problem (Resende, et al., 2010), set packing problem (Delorme, Gandibleux & Rodriguez, 2004), generalized quadratic assignment problem (Mateus, Resende & Silva, 2011), multi-plant capacitated lot sizing problem (Nascimento, Resende & Toledo, 2010) and set k-covering problem (Pessoa, et al., 2012). Festa and Resende (2011) give an overview of GRASP and its enhancements including the PR strategy.

In addition to GRASP, PR was first applied in GA to implement a progressive crossover operation by Ribeiro and Vianna (2003). Ribeiroa and Vianna (2009) extended their proposal and developed a better implementation. Andrade and Resende (2007) showed that a GRASP with evolutionary PR finds solutions faster than a heuristic based on GRASP with PR as well as one based on pure GRASP. The multiple-level warehouse layout problem (Zhang & Lai, 2006) and the minimum tardiness permutation flowshop problem (Vallada & Ruiz, 2010) are among other problems successfully solved by GAs with PR.

Based on the adaptive memory and responsive strategy elements of SS and PR, Yin, et al. (2010) created a combination of PSO and SS/PR to produce a Cyber Swarm Algorithm that proves more effective than standard PSO. Applied to the challenge of finding global minima for continuous nonlinear functions, the Cyber Swarm Algorithm was able to obtain better solutions to a well known set of benchmark functions.

There are many other successful hybrid applications in which PR is used to add a memory mechanism by integrating it into other algorithms, including TS (Armentano, et al., 2011; Nasiri & Kianfar, 2012; Vogt, Poojari & Beasley, 2007), variable neighborhood search (Wang & Tang, 2009), SS (Nasiri and Kianfar, 2011; Ranjbar, Reyck & Kianfar, 2009), ACO (Liu & Liu, 2011), and memetic algorithms (Jaszkiewicz & Zielniewicz, 2009).

Martí, Montes and El-Fallahi (2005) implemented TS methodology coupled with PR for function approximation and obtained the best solutions in terms of quality. El-Fallahi, Martí and Lasdon (2006) proposed a PR implementation to solve the neural network training problem. Their experimentation showed that the proposed procedure can compete with the best-known algorithms in terms of solution quality, consuming a reasonable computational effort.

PR has been applied in connection with different metaheuristics as a combination method, mainly applied to combinatorial optimization problems, but also in the context of continuous optimization (Jaeggi, et al., 2008). A multiobjective combinatorial optimization is another research field for the researchers who have applied the PR approach (Beausoleil, Baldoquin & Montejo, 2008; Pacheco & Martí, 2006). Plateau, Tachat and Tolla (2002) applied PR in combining the solutions encountered in their hybrid search interior point methods and metaheuristics for 0-1 programming.

Recent PR approaches have been developed to solve the large-scale global optimization (Duarte, Martí & Gortazar, 2011). Ribeiro and Resende (2012) reviewed PR intensification methods for stochastic local search algorithms. Detailed explanations of

PR is presented by Glover (1999) and Glover, Laguna and Marti (2000). A survey reporting on advanced PR strategies can be found in Resende and Ribeiro (2005).

7.2 Path Relinking Algorithm

From the standpoint of metaheuristic classification, it has been mentioned that Scatter Search (SS) is an evolutionary algorithm that constructs solutions by combining others. Features of SS are also captured in the PR algorithm. Both approaches originally stem from strategies of combining decision rules and constraints in the context of integer programming (Glover, Laguna & Martí, 2003). The basic idea behind the PR is to reinterpret the linear combinations of points in the Euclidean space as paths between and beyond solutions in the neighborhood (Talbi, 2009).

The approach is named PR because it generates a path between solutions linked by a series of moves during a search to incorporate attributes of the guiding solution while recording the objective function values (Glover & Laguna, 1997). The PR approach generates new solutions by exploring trajectories connecting the initiating solution and the guiding solution. While following the path from the initiating towards the guiding solution the high-quality solutions are created by selecting moves with "good" attributes contained in the guiding solution (Glover, Laguna & Martí, 2003). At each iteration, the best move in terms of the objective function and decreasing the distance between the two solutions is selected. This is repeated until the distance is equal to 0 at which point the best solution found in the trajectory is returned by the algorithm.

PR is different from local search approaches in many ways: the path between initial and guiding solutions is directed by the criterion of incorporating attributes of the

guiding solution, not by local attraction. This feature helps PR reach some solutions that would not be found by a "locally myopic" search (Glover, Laguna & Martí, 2003). The relinked path may also provide fertile starting points for creating neighborhoods which may include high quality solutions.

Glover and Laguna (1997) suggested PR as an approach to integrate intensification and diversification strategies in the context of TS. PR can be used to diversify or intensify the search, depending on the path generation and the choice of the initial and guiding solutions (Gendreau & Potvin, 2007). In PR, for each pair of initial and guiding solutions there exist different alternatives in selecting the starting and the target solutions:

- Forward: The worst of both solutions is used as the starting solution.
- Backward: The better of both solutions is used as the starting solution. Since the starting solution's neighborhood is more explored than that of the target solution, the backward strategy is, in general, better than the forward one.
- Backward and forward relinking: Two paths are constructed in parallel, using alternatively both solutions as the starting and the target solutions.
- Mixed relinking: Two paths are constructed in parallel from both solutions but the guiding solution is an intermediate solution at the same distance from both solutions.

Besides these path forms, there are also multiparent path generation possibilities in PR by considering the combined attributes of a set of guiding elite solutions. In the multiparent path generation, proper weights are given to these attributes in order to determine which directions are given higher priority. Building a set of elite solutions as

multiparents creates a constructive approach in creating new solutions in PR. In this case, the initial solution begins as a partial solution or as a null solution, where some of the components of the solutions, i. e. items in 0-1 MKP, are not yet assigned. The constructive neighborhood structure allows the initial solution to move toward the guiding solutions by a neighborhood path introducing components contained in the set of guiding solutions based on their attractiveness (Glover, Laguna & Martí, 2003).

The PR approach can also utilize a powerful optimization technique, constraint relaxation, to increase the possibility of obtaining high quality solutions by enlarging the search space. Constraint relaxation as an attractive strategy that creates a larger search space can be implemented by dropping some constraints and adding weighted penalties to the objective function for the constraint violations. In the 0-1 MKP, the constraint relaxation method can be applied to PR by allowing solutions exceeding the capacity of one or more knapsacks. In this case, penalty weights can be determined systematically by leading the search to cross the feasibility boundary of the search space. This technique is known as strategic oscillation, introduced in Glover (1977) and used in several successful TS algorithms. Strategic oscillation requires defining an oscillation, or feasibility boundary, and when the algorithm reaches the feasibility boundary, it continues the search beyond the boundary before turning around. Repeating this process creates an oscillatory search pattern. It is possible to adjust the amplitude of the oscillation; e. g. tight oscillations favor a more thorough search around the boundary (Gendreau & Potvin, 2007). This method, also known as the tunneling strategy, is protected against the possibility of becoming "lost" in an infeasible region, since feasibility evidently must be recovered when the guiding solution is reached (Glover, Laguna & Martí, 2003).

7.3 Meta-RaPS PR Algorithm

In the Meta-RaPS PR algorithm, the improved solution found at the current iteration can be accepted as the initial solution, and the best improved solution found so far as the guide solution. To follow the PR process, the initial and guide solutions are first coded in a binary string. The positions containing the same numbers in the initial and guide solutions are identified to keep their states and the numbers in the remaining positions are changed in a systematic way to create the neighborhood. The neighbor with the maximum profit is selected to build the path. At each step, the solutions become more similar to the guide solution and more different from the initial solution. While processing, the solution found is replaced with the best improved solution only if it is better than the best improved solution.

For example, considering a 4-item 0-1 MKP problem, if items 3 and 4 are selected for the initial solution, and items 1, 2 and 4 for the guide solution, they will be coded as (0 0 1 1) and (1 1 0 1), respectively. Note that initial and guide solutions share only one item with the same state at the same position. The states of items in the other positions are switched from selected (1) to not selected (0), or not selected (0) to selected (1) to obtain the following neighbors: (1 0 1 1), (0 1 1 1) and (0 0 0 1). The best neighbor, i.e. the one with the maximum profit, is selected as the new initial solution, which is now closer to the guide solution, having two items at the same position. This process is followed until the initial and guide solutions are totally identical. Table 32 summarizes the PR transforming process from the initial to guide solutions.

Table 32. Meta-RaPS PR Process

Initial	Guide	Neighbors		
0011	1101	1011*	0111	0001
1011	1101	1111	1001*	
1001	1101	1101*		
1101	1101			

The PR phase of the Meta-RaPS PR algorithm is not executed at the first iteration because the best improved solution to serve as the guide solution is not constituted yet. The Meta-RaPS PR pseudo code is shown in Figure 15.

```
For iteration ≤ I
    Apply Meta-RaPS rules to produce ImprovedSolution &
BestImprovedSolution
    Assign ImprovedSolution as InitialSolution
    Assign BestImprovedSolution as GuideSolution
    While (InitialSolution ≠ GuideSolution)
        Create CandidateSolutions
        Assign BestCandidateSolution as PathRelinkingSolution
        If PathRelinkingSolution > BestImprovedSolution then
            Assign PathRelinkingSolution as BestImprovedSolution
        Assign PathRelinkingSolution as InitialSolution
    End While
End For
Report BestImprovedSolution
```

Figure 15. Meta-RaPS PR Pseudo Code

7.4 Meta-RaPS PR for Small and Medium 0-1 MKP Instances

In the construction phase of Meta-RaPS, a solution for the 0-1 MKP is built by repeatedly adding feasible items to the current solution (partial solution) in the order based on their priority rules until the stopping criterion is satisfied. As in the previous chapters, there are two versions of Meta-RaPS PR depending on the greedy rule used. While Meta-RaPS PR-G is uses DGR to obtain priority rules in selecting items, Meta-RaPS PR-W is the independent version that considers the weights of items only. The detailed results of both algorithms are summarized in Table 33.

Table 33. Meta-RaPS PR Results for Small/Medium 0-1 MKP Instances

	Deviation%			Iteration	Time	Optimum	
Version	**IMean**	**IBest**	**CMean**	**Number**	**(Sec.)**	**Frequency**	**Instance**
Meta-RaPS PR-W	0.132	0.048	1.884	1851	588.57	7.60	49
Meta-RaPS PR-G	0.001	0.000	1.282	480	47.93	9.76	55
Average	**0.067**	**0.024**	**1.583**	**1166**	**318.25**	**8.68**	**52.00**
Std.Dev.	**0.093**	**0.034**	**0.426**	**969**	**382.29**	**1.53**	**4.24**

Meta-RaPS PR-W could find the optimum values for 49 of 55 instances. Their average deviation percentage from optimum/best solutions for the CSs and ISs are 1.884% and 0.132%, respectively. Meta-RaPS PR-W obtained the optimum solutions on average 7.6 out of 10 times. The average time and iterations to solve the instances are 589 and 1,851, respectively.

On the other hand, the Meta-RaPS PR-G approach could solve all small and medium instances, and found the optimum/best solutions (9.8 out of 10) run on average

for all instances. The average deviation percentages of CSs and ISs reached by the proposed algorithm were 1.282% and 0.001%, respectively. Meta-RaPS PR-G obtained these results in an average of 48 seconds and 480 iterations, respectively.

Because of the higher performance of Meta-RaPS PR-G over W version, it is accepted as the main version of Meta-RaPS PR. To reveal the contribution of the PR to Meta-RaPS, the number of optimum/best solutions found in the construction, improvement and PR phases are tracked for each instance. Since it is observed in the initial analysis that the role of PR is getting more important with the increasing number of items and knapsacks, the instances are put in the order of size, which is defined here as the product of the number of items and number of knapsacks.

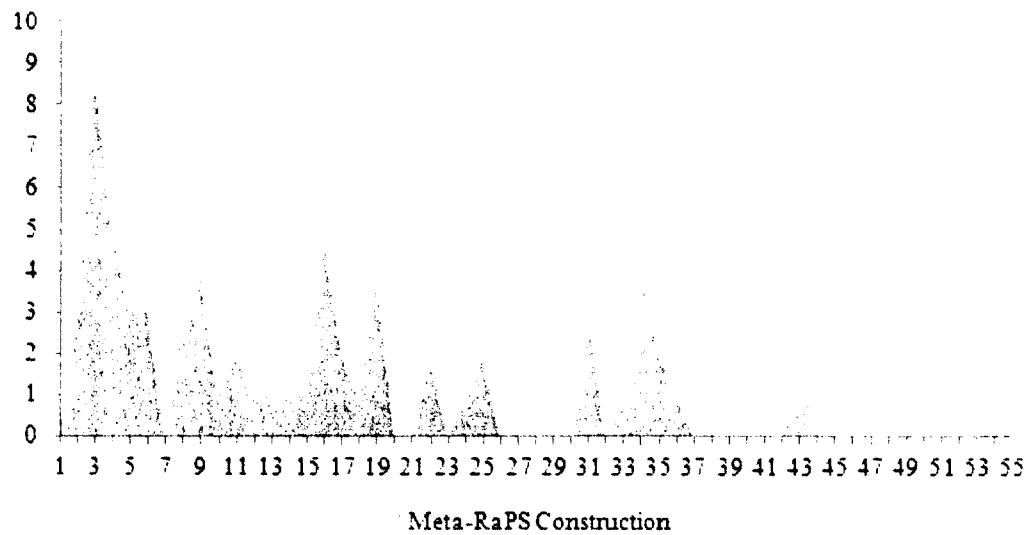

Figure 16. The Number of Optimum Solutions Found in 10 Replicates of Meta-RaPS PR Construction Phase for 55 Small/Medium Instances

Figure 16 shows the distribution of the number of optimum solutions found in the construction phase. For the instances with lower size, Meta-RaPS could find optimum or best solutions, and for the larger instances the chance of reaching to optimum or best solutions is decreases. The distributions of the number of optimum/best solutions found in the improvement and PR phases show the efficiency of the PR algorithm (Figure 17). Especially for the larger instances, the role of PR in the proposed algorithm is clear; its share of number of optimum or best solutions found in 10 runs for each instance increased.

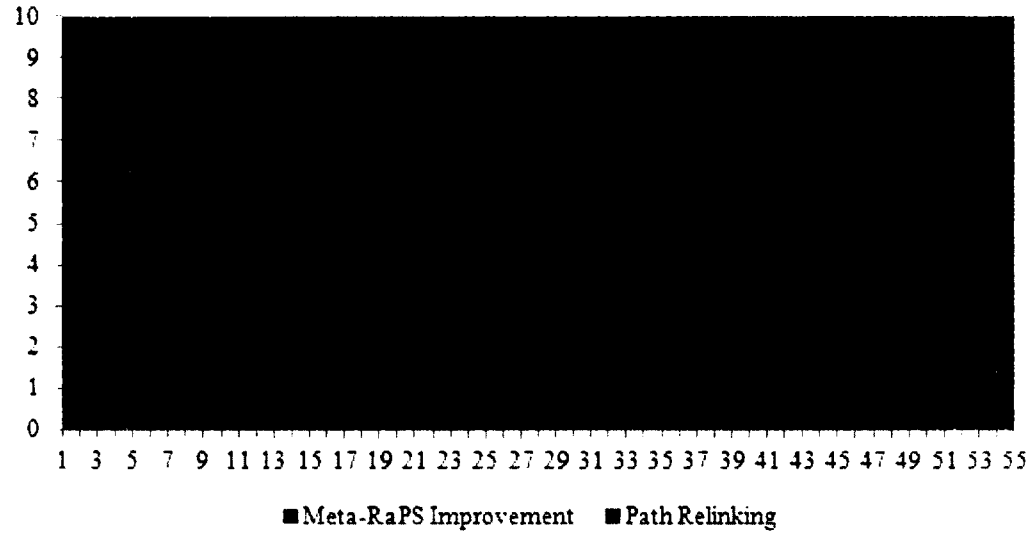

Figure 17. The Distribution of Best Solutions Found in 10 Replicates of Meta-RaPS PR Improvement and PR phases for 55 Small/Medium Instances

To look closer the distribution of best solutions found in the improvement and PR phases, their best solutions and trendlines found in 10 replicates were depicted in Figures 18a and b. respectively. As seen from the trendlines in these figures, increasing the size

of the instances makes decrease the share of the improvement phase in producing the best solutions in 10 replicates; and increase the PR phases share. In these figures, the method of polynomial trendlines (order 3) were used to observe the predictions since the polynomial trendline produced larger R^2 values than other methods did, such as linear, logarithmic, or exponential trendlines.

Recall that besides the parameter of number of iterations (I), there is another stopping criterion, which is when the deviation percentage is equal to 0. For the instances with smaller size, Meta-RaPS can find optimum solutions, and stops the solution process before the algorithm reaches the PR phase. This is the reason why the PR phase seems to not produce optimal or best solutions for these instances. However, in solving large instances, Meta-RaPS is expected to call the PR phase to obtain better results.

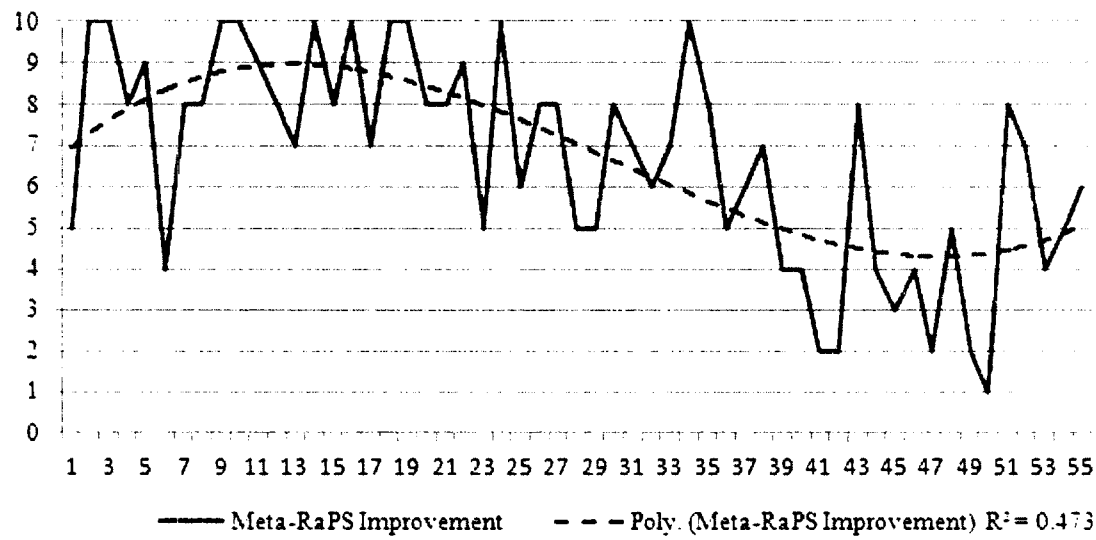

Figure 18a. Trendline of Best Solutions Found in 10 Replicates by Improvement Phase of Meta-RaPS PR for 55 Small/Medium Instances

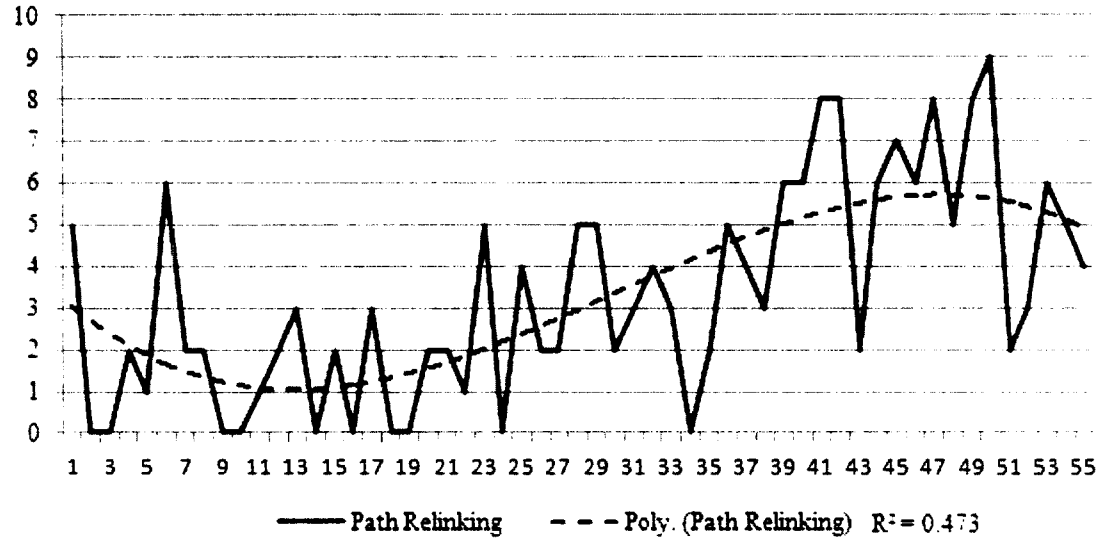

Figure 18b. Trendline of Best Solutions Found in 10 Replicates by PR Phase of Meta-RaPS PR for 55 Small/Medium Instances

7.5 Meta-RaPS PR for Large 0-1 MKP Instances

With the same parameter setting used in Meta-RaPS EDA and Q versions, Meta-RaPS PR was run to solve the first three sets of large size 0-1 MKP instances. As shown in Table 34, the overall average deviations from optimum/best solution found was 0.142% in an average of 21 minutes and 2,988 iterations. Meta-RaPS PR algorithm obtained the optimum values for 28, 22 and 14 instances of the 100 items with 5, 10 and 30 knapsacks, respectively. The average optimum instance was 21.3 of 30 instances for the first three sets. Meta-RaPS PR could reach the optimum/best results in 4.9 of 10 replications. The best average performance of the Meta-RaPS PR, i. e. the best average deviations percentage, for all instances was 0.061, and the overall average deviation percentage for CSs is 0.640.

Table 34. Meta-RaPS PR Solution for Large 0-1 MKP Instances

	Deviation%			Iteration	Time	Optimum	
Instance Set	**IMean**	**IBest**	**CMean**	**Number**	**(Min.)**	**Frequency**	**Instance**
100x5	0.031	0.008	0.353	2321	14.44	7.27	28
100x10	0.155	0.080	0.650	2819	27.32	5.17	22
100x30	0.252	0.095	0.917	3824	86. 17	2.17	14
Average	0.146	0.061	0.640	2988	20.88	4.87	21.33
Std.Dev.	0.111	0.047	0.282	766	9.11	2.56	7.02

CHAPTER 8

INCORPORATING ADAPTIVE PARAMETER TUNING INTO META-RAPS

Adaptive parameter setting is another form of learning in metaheuristics since it requires the algorithm to memorize and learn the best parameters that could create high quality solutions in the search history. Different from other memory and learning approaches proposed in previous chapters, incorporating an adaptive parameter setting mechanism into Meta-RaPS is thought as the next task to accomplish in the way of creating a smart algorithm.

Although metaheuristics are found to be very effective and efficient for optimization problems, they are sensitive to the values their parameters take and therefore it is very important to run them with the appropriate parameter setting(s) to reach high quality solutions. Parameter tuning is also critical to make the algorithm intensify or diversify its search process. Balancing between intensification and diversification in the search space is a key factor to reach fertile search areas and avoid premature convergence (Wong, 2008).

Parameter tuning often requires either a deep knowledge of the problem structure, or trial and error algorithms with long tuning experiments. There is no unique parameter setting for metaheuristics as they are applied to solve different problems. There is anecdotal evidence that in designing and testing of a new metaheuristic, about 10% of the total time is allocated to development, and the remaining 90% of the time is spent on tuning parameters (Belarmino & Laguna, 2006).

A systematic and simple way to determine parameters would be to increase algorithmic efficiency (Battarra, et al., 2012). A powerful approach to tuning parameters is by controlling them throughout the search process, which is generally called an adaptive, reactive, or self-tuning metaheuristics. These metaheuristics utilize feedback information obtained during the search to perform a learning process of the parameter combination (Alabas-Uslub & Dengiz, 2011). Birattari (2009) reported that adaptive parameter tuning approach is particularly appealing when solving one single instance, typically large and complex.

8.1 Literature Review

Self-adaptive heuristics are achieved for evolutionary algorithms earlier than local search based algorithms (Alabas-Uslub & Dengiz, 2011). The development of parameter adaptation mechanisms in EAs began in 1967, when Reed, et al. (1967) learned to play poker with an EA, and Rosenberg (1967) proposed to adapt the probability for applying crossover. Weinberg (1970) and Mercer and Sampson (1978) first introduced meta-evolutionary approaches where an outer EA mechanism controls the parameters of an inner mechanism that solves the problem. The term self-adaptation is commonly associated with the self-adaptation of mutative step sizes for Evolutionary Strategies (ES) like those introduced by Rechenberg (1973) and by Schwefel (1974). After ES, Fogel, et al. (1991) introduced self-adaptation to Evolutionary Programming (EP). Kramer (2010) reported that for binary coded EAs, self-adaptation has not grown to a standard method, only few theoretical investigations of self-adaptation exist; mostly on continuous search

domains and the analysis of mutation strengths in ES. According to Beyer and Schwefel (2002), the analysis of EAs, including the mutation control part, is a difficult task.

The idea of considering GAs for tuning the parameters of the heuristics was first introduced by Golden, et al. (1998) in their two-phase procedure. During the first phase, the algorithm was trained on a small set of representative problem instances by determining a parameter vector that guarantees a good performance. In the second phase, the generated parameter vectors were linearly combined into an overall vector with the weights of the linear combination determined by the genetic procedure, so that the best overall performance is reached (Battarra, et al., 2012). Pepper, et al. (2002) use a similar technique to set the parameter of an annealing based heuristic for the travelling salesman problem (TSP), and Chandran, et al. (2003) further analyze the possibility of applying a genetic parametric search procedure by introducing a simpler single-stage procedure.

Kivijarvi, et al. (2003) proposed a self-adaptive GA for the clustering problem. Their algorithm gave very high quality results for hard problem instances. Binkley and Hagiwara (2007) introduced two different EA algorithms: a self-adaptive parallel recombinative simulated annealing algorithm, and a self-adaptive GA. They informed that the results were best in the published literature and the self-adaptive GA outperformed the fixed parameter GAs on the larger problems. In their papers, Battarra, et al. (2008, 2012) proposed a single-stage GA-based procedure for tuning the parametric Clarke and Wright (CW) heuristic and the Esau and Williams (EW) heuristic. Birattari (2009) adopts a machine learning perspective to the tuning problem of metaheuristics and develops a GA for the tuning.

Castellani, et al. (2007) presented an optimization technique to automatically select a set of control parameters for a Markov random field (MRF) based on the reactive tabu search strategy. Prais and Ribeiro (2000) proposed a new procedure, Reactive GRASP, in which the basic parameter that defines the restrictiveness of the candidate list during the construction phase is self-adjusted according to the quality of the solutions previously found. Their approach was robust and does not require calibration efforts. The Reactive Search (RS) method was applied to GRASP algorithm (Gomes, et al., 2001; Junior, et al., 2008; Usberti, et al., 2011). Hepdogan, et al. (2008) applied dynamic parameter setting of Meta-RaPS based on both RS and GA, called a Non-Parametric Based Genetic Algorithm (NPGA). NPGA compares parameter settings with each other to determine if they are statistically better than each other by using non-parametric methods. Comparing the two dynamic parameter setting techniques considered, they reported that NPGA performed better than RS.

Ide and Yasuda (2005) proposed an adaptive search algorithm for PSO in both continuous and discrete domains, in which the parameters are tuned to the problem structure at every search point by updating of settings based on comprehension of the agent's current state.

Favuzza, et al. (2006) have successfully shown that they have used dynamic parameter tuning as a strategy to balance intensification and diversification in ACO. Wong (2008) produced a review on researches related to parameter tuning as a strategy to balance intensification and diversification in ACO. Anghinolfi, et al. (2008) proposed a self-adaptive ACO algorithm that exploits a parameter adaptation mechanism to reduce the requirement of a preliminary parameter tuning. They tested the proposed approach on

the single machine total tardiness scheduling problem with sequence-dependent setups, and could improve the benchmark best known results. Tamilarasi (2010) presented an ACO method to solve the job shop scheduling problems with step pheromone updating strategy based on statistical analysis. The author reported that once the parameters are properly tuned, the algorithm converges satisfactory.

Abraham (2004) proposed a framework for optimization of artificial neural networks, where learning algorithm and its parameters are adapted according to the problem. Ramos, et al. (2005) proposed using the logistic regression approach in tuning the parameters of an EA. Logistic regression describes the relationship between the categorical response variable and one or more continuous or categorical explanatory variables. Although it requires additional computational effort to tune the parameters, the new algorithm showed that when there is evidence of the goodness of fit of the model, the technique can direct the parameter setting, and provide data to support conclusions about the best policy to be adopted.

Alabas-Uslub and Dengiz (2011) developed another heuristic algorithm, named self-adaptive local search (SALS), with a self-adaptive mechanism for solving the classical vehicle routing problem allowing for the escape from the difficulty of parameter optimization. The proposed heuristic had only one generic parameter, called the acceptance parameter, calculated and updated self-adaptively throughout the search process to improve the effectiveness of the algorithm using the response surface information which comes from the problem and the performance measure of the algorithm. Besides its simplicity, SALS also provided qualified solutions to well-known benchmark problems from the VRP literature within reasonable amount of computation

times. The same approach was also applied for the multi-objective vehicle routing problem (Alabas-Uslub, 2008) and flow-shop scheduling problem (Dengiz, et al., 2009). Ries, et al. (2012) proposed an instance-specific method for parameter tuning, called IPTS. IPTS created a link between instance characteristics and the decision maker's preference with respect to the solution quality – computational time tradeoff, to algorithm-specific parameter values.

There have also been researches about meta-learning approaches for parameter setting. Chen, et al. (2002) used inductive meta-learning and clustering to tune parameters and choose the algorithm. Cail, et al. (2006) proposed an automatic parameter tuning method based on machine learning. Soares, et al. (2004) presented a meta-learning approach to parameter setting that exploits information about past performance of different settings. Sikora (2008) used a simple meta-learning algorithm to learn the temperature parameter of the Softmax reinforcement-learning algorithm.

Eiben, et al. (1999) presented a study to classify parameter control methods for EAs and survey various forms of control methods. De Jong (2007) gives a detailed overview of parameter setting overviewing 30 years of research in this area. Kramer (2010) produced an extensive survey and a textbook (2008) about evolutionary self-adaptation of operators and strategy parameters. Birattari (2009) created another comprehensive textbook on tuning metaheuristics in the machine learning perspective.

These successful applications of adaptive parameter tuning in metaheuristics support our belief in creating a promising method in which the algorithm can adaptively tune the parameters of Meta-RaPS. Therefore the approach used here will be focused on the methods of adaptive parameter tuning.

8.2 Adaptive Parameter Tuning

Although it is accepted that appropriate parameter settings can lead a search in the right direction, they require knowledge about the problem structure. For many black-box optimization problems there is no knowledge about the search space. On the other hand, the best parameter setting usually depends on the application area, size or input data of the problem for each of the problem instances (Alabas-Uslub & Dengiz, 2011). In these cases it would be very convenient if parameters of algorithms were tuned autonomously for each problem.

Eiben, et al. (1999) presented two main types of parameter settings techniques for evolutionary algorithms: parameters that have to be tuned before or controlled during the run of the optimization algorithm. In their taxonomy, parameter tuning can be executed by hand; design of experiments (DOE) or meta-evolution; and parameter control can be reached by deterministic, adaptive and self-adaptive techniques. In the case of tuning by hand, the efficiency of the parameter setting only depends on human experience; however it may not be the optimal parameter setting. DOE requires a statistical analysis of experiments, i. e. trial solutions executed with different parameter set by a detailed experimental plan. In meta-evolutionary algorithms, also known as nested evolutionary algorithms, an outer optimization algorithm tunes the parameters of an embedded algorithm (Rechenberg, 1994).

Eiben, et al. (1999) called the change of parameters during the run as online parameter control due to the fact that the conditions of the fitness landscape can change during the optimization process (Kramer, 2010). In the deterministic parameter control, parameters are changed depending on some fixed factors, e. g. the number of generations

in EAs. Adaptive parameter control methods use feedback from the search to determine magnitude and direction of the parameter change under the rules defined by the practitioner. An example for an adaptive control can be the 1/5-th success rule for the mutation strengths in EA by Rechenberg (1973) where the step sizes are increased, if the success ratio is higher than 1/5-th to allow faster progress and is decreased if the success rate is lower than 1/5-th.

Self-adaptation is a well-known concept meaning that the algorithm is capable of adapting itself totally autonomously. Self-adaptation is based on the theory that good solutions more likely result from good parameter settings than from bad ones. These good settings of parameters will have a high probability of being selected by the algorithm while processing. According to Kramer (2010), a necessary condition for a successful self-adaptation is the existence of a tight link between parameters and fitness; i.e., if the quality of the search process heavily depends on a particular setting of parameters. Self-adaptive parameters are also known as endogenous, i.e., evolvable, in contrast to exogenous parameters, which are kept constant during the optimization run (Beyer & Schwefel, 2002). Self-adaptation plays the role of the stochastic online control toward a parameter-free optimization metaheuristic (Kramer, 2010).

The Reactive Search (RS) is another powerful method in setting parameters which uses feedback from the metaheuristics. RS incorporates a history-based adaptive procedure in the search to determine the values of parameters. This approach investigates a variety of parameter settings while the algorithm is running and determines the probabilities of selecting each parameter setting based on their fitness values, i. e. higher probabilities for the parameters which lead to the best solutions.

The pioneering research in developing a self-adaptive mechanism for the local search based metaheuristics created the reactive tabu search by Battiti and Tecchiolli (1994). The most critical parameter in TS usually is the tabu list size which balances between intensification and diversification strategies. Given the fixed size of the tabu list, the search might be trapped in a cycle of length greater than the size list. In order to cope with this drawback, the reactive tabu search dynamically adjusts the tabu list size by memorizing the history of the search process to determine the probability of selecting each parameter setting for future iterations.

Discovering the relationships between the parameters and the search trajectories has a major impact in metaheuristics to reach the best solutions. The online parameter tuning methods, particularly adaptive/self-adaptive methods, are among best candidates that can make parameters evolve to their best settings. This fact encourages tuning the parameters of Meta-RaPS adaptively.

8.3 Meta-RaPS Adaptive Parameter (AP) Algorithm

To tune the parameters online, i.e. change adaptively, Meta-RaPS algorithm needs a mechanism to memorize and learn the effects of different parameter settings on the solution process. This mechanism can be formed via a parameter memory matrix, similar to the idea presented in EDA.

The parameter memory matrix for Meta-RaPS is created for the parameters *priority* and *restriction*, containing 9 levels between 0.1 and 0.9 with increments of 0.1 for each parameter. The parameter *improvement* is accepted 0.1 according to the results of D-Optimal design applied in Section 4.4. Thus, 81 (= 9 x 9) different parameter

settings can be attempted in solving the 0-1 MKP instances. These parameter settings are then applied in solving the instances and their solution values are recorded in the cells representing the parameter settings of *priority* and *restriction*. Once the parameter settings in the cells are assigned, the solutions can be generated randomly or by applying a greedy rule in both the parameter memory matrix and the solution process. The Meta-RaPS with online tuning, or adaptively changing parameters will be named a Meta-RaPS AP (Adaptive Parameter).

The way of employing the parameter settings represented in the cells forms different versions of Meta-RaPS AP. Each parameter setting can be selected randomly to solve the instances, and in this case it cannot be guaranteed that the number of times each cell selected is equally likely to be filled in. To prevent this, the algorithm can be pushed to select each cell the same number of times. On the other hand, the best or average solution values obtained by using the corresponding parameter settings can be used to update the values in the cells of the parameter memory matrix for the sake of producing different results. All these features have created four different versions of Meta-RaPS AP, as shown in Table 35.

Table 35. Meta-RaPS AP-G Solutions for Small/Medium 0-1 MKP Instances

	Update Method	
Update Chance	**Best Value**	**Average Value**
Equal	Version - 1	Version - 2
Random	Version - 3	Version - 4

Since the DGR-based versions of Meta-RaPS have created better solutions than randomly generated versions in the previous chapters, the Meta-RaPS AP with DGR, named as Meta-RaPS AP-G, was employed to specify the best approach in Table 20 in forming the parameter memory matrix. Depending on the results, the version of Meta-RaPS AP-R, where the parameter memory matrix and the solutions are produced randomly, will be investigated with this best approach.

The results showing their performances in solving the small/medium 0-1 MKP instances are summarized in Table 36. Version 3, where the parameter settings in cells are applied randomly and each cell is updated by taking the best values, has created the best solutions among all four versions. Table 37 presents the average values for the parameters used in each version of Meta-RaPS AP-G.

Table 36. Meta-RaPS AP-G Solutions for Small/Medium 0-1 MKP Instances

	Deviation%			Iteration	Time	Optimum	
Version	IMean	IBest	CMean	Number	(Sec.)	Frequency	Instance
1	0.036	0.007	0.691	1410	321.72	8.25	52
2	0.036	0.008	1.702	1679	473.04	8.24	52
3	0.012	0.002	0.107	506	257.34	9.55	54
4	0.076	0.032	0.727	412	96.27	8.58	50
Average	**0.040**	**0.012**	**0.807**	**1002**	**287.09**	**8.66**	**52.00**
Std.Dev.	**0.027**	**0.013**	**0.661**	**637**	**156.07**	**0.62**	**1.63**

Table 37. Parameters of Meta-RaPS AP-G for Small/Medium 0-1 MKP Instances

Versions	Parameter	
	Priority	Restriction
1	0.100	0.100
2	0.100	0.100
3	0.285	0.341
4	0.876	0.522

```
While (Parameter Memory Matrix not converged)
    Initialize Parameter Memory Matrix with zeros
    Select randomly a cell representing a parameter setting (p,r)
    Generate a solution by Meta-RaPS using the parameter setting (p,r)
    If GeneratedSolution(p,r) > ParameterMemorySolution(p,r) then
       Assign GeneratedSolution(p,r) as ParameterMemorySolution(p,r)
End While
For iteration ≤ I
    Select best ParameterMemorySolution(p,r) in Parameter Memory Matrix
    Accept (p,r) as parameter setting for current iteration
    Apply Meta-RaPS rules to produce ImprovedSolution &
    BestImprovedSolution
    If ImprovedSolution > ParameterMemorySolution(p,r) then
       Assign ImprovedSolution as ParameterMemorySolution(p,r)
End For
Report BestImprovedSolution
```

Figure 19. Meta-RaPS AP Pseudo Code

Since Version 3 could produced the best results among all four AP versions, Meta-RaPS AP will be designed by creating a memory set for which the parameter setting in each cell is applied randomly and updated by taking the best values of all replications. The pseudo code of Meta-RaPS AP is presented in Figure 19.

8.4 Meta-RaPS AP for Small and Medium 0-1 MKP Instances

After the analysis to obtain the best algorithm in AP approach, Version 3 was accepted as Meta-RaPS AP-G. To design the independent version of the Meta-RaPS AP, i.e. without using any greedy rule, the proposed algorithm gives the priorities simply based on their weights. This version is named Meta-RaPS AW, and applied to the small/medium 0-1 MKP instances. The results of both AP algorithms are presented in Table 38.

Table 38. Meta-RaPS AP Results for Small/Medium 0-1 MKP Instances

	Deviation%			Iteration	Time	Optimum	
Version	**IMean**	**IBest**	**CMean**	**Number**	**(Sec.)**	**Frequency**	**Instance**
Meta-RaPS AP-W	0.242	0.091	1.651	1979	637.29	1.91	46
Meta-RaPS AP-G	0.012	0.002	0.107	506	257.34	9.55	54
Average	**0.127**	**0.047**	**0.879**	**1243**	**447.32**	**5.73**	**50.00**
Std.Dev.	**0.163**	**0.063**	**1.092**	**1042**	**268.67**	**5.40**	**5.66**

Meta-RaPS AP-G could reach the optimum/best solutions for 54 of 55 instances. PETERSEN6 is the only instance whose optimum value could not be obtained by the algorithm. The mean deviation percentage of the constructed and improvement solutions

is 0.107 and 0.012, respectively. The average time needed to solve the instances is around 257 seconds, and its average iteration is 506. Meta-RaPS AP-W which does not use any greedy rule, could reach the optimum solutions for only 46 instances, and its average deviation and time are much higher than Meta-RaPS AP-G. Thus, Meta-RaPS AP-G version was accepted as Meta-RaPS AP algorithm to apply in larger instances.

In other Meta-RaPS applications, the parameters are tuned before the solution process begins, and the same settings are applied to all instances. However, in the AP tuning versions of Meta-RaPS, the goal is to make the algorithm change the parameters adaptively for each instance, and also in each iteration to reach best parameter settings for the instance, in other words tune the parameters.

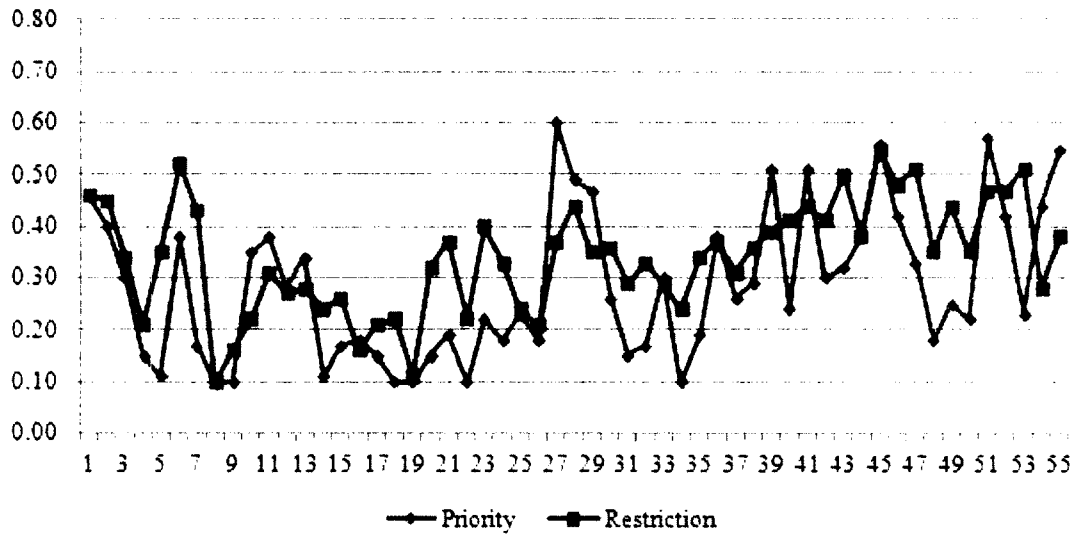

Figure 20. Trend of Parameters for Instances in Meta-RaPS AP-G

The changing process of the parameters *priority* and *restriction* can be observed in Figure 20 created the Meta-RaPS AP-G. To be able to observe the trend of the

parameters depending on their number of items and knapsacks, the instances are ordered to their instance difficulty as defined in the Meta-RaPS PR. Roughly speaking, it can be argued that the values of the parameters *priority* and *restriction* increase with the number of items and knapsacks of the instances.

8.5 Meta-RaPS AP for Large 0-1 MKP Instances

The proposed adaptive algorithm is also applied in large size 0-1 MKP instances, and its observed performance is summarized in Table 39. The average deviation percentage of the algorithm for the first three sets is 0.115 with the average number of iterations of 2081. Meta-RaPS AP produced these results in average of 44 minutes, respectively. On average, Meta-RaPS AP could find optimal solutions for 22 of 30 instances, in 5.6 of 10 replications.

Table 39. Meta-RaPS AP Solutions for Large 0-1 MKP Instances

	Deviation%			**Iteration**	**Time**	**Optimum**	
Instance Set	**IMean**	**IBest**	**CMean**	**Number**	**(Min.)**	**Frequency**	**Instance**
100x5	0.019	0.005	0.081	1329	25.34	7.90	28
100x10	0.098	0.050	0.501	2215	20.72	6.17	23
100x30	0.227	0.083	0.775	2698	84.70	2.67	15
Average	**0.115**	**0.046**	**0.452**	**2081**	**43.59**	**5.59**	**22.00**
Std.Dev.	**0.105**	**0.039**	**0.350**	**694**	**35.68**	**2.67**	**6.56**

The averages of parameters found by Meta-RaPS AP for each of three sets of large instances are shown in Table 40. The average values of parameters *priority* and *restriction* are fairly close in these sets.

Table 40. Parameters of Meta-RaPS AP for Large 0-1 MKP Instances

Instance Set	**Parameter**	
	Priority	**Restriction**
100x5	0.579	0.641
100x10	0.607	0.626
100x30	0.610	0.641
Average	**0.598**	**0.636**
Std.Dev.	**0.017**	**0.009**

CHAPTER 9

REDESIGNING META-RAPS

Metaheuristics can be observed as the repetition of the two main phases; generation of solutions and its improvement by local search (Ibaraki, et al., 2005). In the first phase, solutions are produced based on the principles of the algorithm, by gradually constructing or forming the whole solution at once. Most of the time, the initial solution is not expected to include the attributes of a high quality solution, thus in the second phase, the algorithm requires improving the initial solution by implementing various types of local search techniques.

Although the best solutions of the algorithms are generated by their improvement phase, there is usually a high computational cost of employing the local search. This critical part of the local search is the computational burden that the practitioners should accept *a priori*. For many applications, local search techniques consume more time than generating the solutions. The criteria for selecting a local search technique may be the expectation that the quality of improved solutions by local search will compensate for its computational cost. Otherwise, metaheuristics would lose one of their most important advantages which is the ability to find good solutions in an acceptable time frame.

The results of the proposed algorithms in the previous chapters also reflect this fact. It can be easily observed that increasing the size of the instances requires more computational time, and in the literature there are other larger instances not solved with the proposed algorithms. It can be assumed that these algorithms do already have high quality solutions, and now the next task will be to obtain them in a shorter time frame.

In the proposed algorithms the total costs come from three main factors: memory and learning mechanisms, Meta-RaPS construction phase and improvement phase. Since the aim of this study was to make Meta-RaPS intelligent by introducing memory and learning mechanisms, this factor cannot be avoided. Meta-RaPS construction phase is another factor needed to create initial solutions and thus cannot be left out. The third factor, Meta-RaPS improvement phase, is the only one that can potentially be reduced or eliminated. However, the qualities of the CSs created by the proposed algorithms are not very good and they need to be improved. Such improvements should be made by the Meta-RaPS PR version instead of local search. While in Meta-RaPS EDA, Q and AP versions, memory and learning happen before running the algorithm by showing the right way in constructing solutions, in the Meta-RaPS PR version, memory and learning begin handling the algorithm after Meta-RaPS has constructed the initial solution. This phenomenon acts like the improvement phase instead of using local search. In Chapter 7, there was important evidence presented in Figures 17 and 18 that more optimum solutions in the Meta-RaPS PR were found by PR than the local search phase when increasing the size of instances. Both of these facts encourage the redesign Meta-RaPS by replacing the local search (or improvement) phase with and the PR approach. In this new Meta-RaPS, the constructed solutions will not be improved by local search techniques; instead, they will gain the "good" attributes of the good solutions by learning.

9.1 Redesigning Meta-RaPS

In Meta-RaPS PR algorithm the basic form of PR was implemented, however, a PR application in a metaheuristic without local search should be more sophisticated.

Based on the information given in Chapter 7, three factors can be accepted to design an effective PR algorithm: feasibility of initial and guide solutions, direction of the path, and selection of the initial solutions for the next path. Besides feasibility, infeasibility can also be accepted for the initial and guide solutions, since the solutions will be eventually feasible when PR is completed. The path can start from the CS and reach to the best solution (BS) or from the BS to the CS. Selection of the initial solutions among the candidates for the next path is the last issue, and can be done by taking the solution with maximum value, randomly or applying a greedy rule.

These options create 12 (= 2 x 2 x 3) different alternatives of the PR approach. To evaluate them, 8 instances whose average deviations from optimal/best solutions are different from zero at least two times for all proposed algorithms are selected in Table 41. For the greedy rule, the parameter *priority* is used to select the initial solution for the next path, i. e. if the random number $\leq$ *priority* then select the candidate solution as next initial solution.

These PR alternatives are applied to solve the 8 instances in 10 replications, and their results are summarized in Table 42. Among all alternatives, Feasible/BS-CS/Greedy PR approach gives best results; however, Infeasible/BS-CS/Greedy PR is very close the best alternative.

Table 43 presents the overall averages of solutions according to the options. The greedy approach in PR gives again the best results, and the solutions obtained in both directions are close to each other.

Table 41. Instances to Evaluate PR Alternatives

#	Name	Items	Knapsacks
1	HP2	35	4
2	PB2	34	4
3	PETERSEN6	39	5
4	PETERSEN7	50	5
5	WEING6	28	2
6	WEING7	105	2
7	WEING8	105	2
8	WEISH18	70	5

Table 42. Summary of Solutions by PR Alternatives

Feasibility	Direction	Criteria	Dev.%	Iteration	Time (Sec.)	#Opt. Rep.
Feasible	CS to BS	Maximum	0.096	4223	3.18	5.25
		Random	0.078	2851	3.32	5.75
		Greedy	0.067	3202	2.37	7.13
	BS to CS	Maximum	0.041	1574	1.98	7.63
		Random	0.055	2212	13.64	6.88
		Greedy	**0.026**	**2262**	**2.09**	**8.25**
Infeasible	CS to BS	Maximum	0.128	4324	3.49	4.88
		Random	0.068	2344	3.38	6.25
		Greedy	0.042	2487	3.12	6.25
	BS to CS	Maximum	0.285	3095	3.38	3.25
		Random	0.069	2371	9.48	6.13
		Greedy	**0.030**	**2616**	**2.71**	**7.13**
		Average	**0.082**	**2797**	**4.34**	**6.23**
		Std. Dev.	**0.070**	**813**	**3.53**	**1.35**

Table 43. Overall Averages of Solutions According to PR Options

Criteria	Dev.%	Iteration	Time (Sec.)	#Opt. Rep.
Feasible	0.061	2721	4.42	6.81
Infeasible	0.104	2873	4.26	4.44
CS to BS	0.080	3239	3.14	5.92
BS to CS	0.084	2355	5.54	6.54
Maximum	0.137	3304	2.99	5.25
Random	0.068	2445	7.45	6.25
Greedy	0.041	2642	2.57	7.19
Average	**0.082**	**2797**	**4.34**	**6.06**
Std. Dev.	**0.031**	**367**	**1.71**	**0.95**

The analyses presented in Tables 42 and 43 indicates that accepting the next initial solution by a greedy rule has the biggest impact on the solution quality. In addition, selecting initial and guide solutions being feasible or infeasible and both directions of the path have also some contributions that should be taken into considerations. Under these circumstances, the new PR algorithm is designed to select the next initial solution by a greedy rule without checking the feasibility of the candidate solutions and its paths will have both directions, i. e. 2-way PR.

By utilizing all lessons learned from the previous chapters, redesigning Meta-RaPS is completed by replacing its improvement phase with this PR approach, and renamed as Meta-RaPS V2 (Version 2). The parameter *improvement%* that decides to perform the improvement phase is renamed as *pathrelinking%* which now decides to perform the PR phase. The pseudo code of Meta-RaPS V2 is presented in Figure 21.

```
For iteration ≤ I
While (feasible solution is not constructed)
      Find priority value for each feasible activity
      Find best priority value
      If rnd() ≤ priority% then
            add item with best priority value to solution
      Else create CandidateList from feasible activities with
            priority values ≥ Limit
            Limit = MinimumPriority +
                  restriction% · (MaximumPriority - MinimumPriority)
            Choose randomly an item from CandidateList and add to solution
End While
Δ = BestConstructedSolution · pathrelinking%
If ConstructedSolution ≥ Δ then apply 2-Way Path Relinking
   For the way from ConstructedSolution to BestSolution;
      Assign ConstructedSolution as InitialSolution
      Assign BestSolution as GuideSolution
      Apply Path Relinking to produce BestSolution
   For the way from BestSolution to ConstructedSolution;
      Assign BestSolution as InitialSolution
      Assign ConstructedSolution as GuideSolution
      Apply Path Relinking to produce BestSolution
End For
Report BestSolution
```

Figure 21. Meta-RaPS V2 Pseudo Code

9.2 Meta-RaPS V2 for Small and Medium 0-1 MKP Instances

To be able to compare the performance of the new Meta-RaPS with the proposed Meta-RaPS versions discussed previously and other algorithms in the literature, it will be applied to both small/medium and large size 0-1 MKP instances. A lesson learned in this research is that the proposed algorithms based on DGR create higher quality solutions than based on greedy rule-free versions; thus, only DGR will be used to create priority rules for the new Meta-RaPS.

Another lesson learned in this research is from the parameter setting area. Meta-RaPS AP has proved how effective the adaptive parameter setting approach can be if applied properly. Therefore, the feedback from the AP applications in Chapter 8 can be used in setting the parameters of the new Meta-RaPS. The average of parameters for the small/medium size instances obtained by the Meta-RaPS AP was presented in Table 37. The overall average values of parameters found by Meta-RaPS AP will now be accepted for the parameters setting of the new Meta-RaPS in Table 44. Table 45 shows the details of the new algorithm for the small/medium 0-1 MKP instances.

Table 44. The New Parameter Setting of Meta-RaPS V2 for Small/Large 0-1 MKP Instances

Parameter	Value
Priority percentage (p)	0.29
Restriction percentage (r)	0.34
Path Relinking percentage (i)	0.10
Number of iterations (I)	10000

Table 45. Meta-RaPS V2 Results for Small/Medium 0-1 MKP Instances

	Deviation%			Iteration	Time	Optimum	
Version	**IMean**	**IBest**	**CMean**	**Number**	**(Sec.)**	**Frequency**	**Instance**
Meta-RaPS V2	0.010	0.000	0.938	416	0.32	9.86	55

9.3 Meta-RaPS V2 for Large 0-1 MKP Instances

As in the case of small/medium size instances, the average parameter settings found by Meta-RaPS AP in Table 40 will be used as the new parameter setting for the new Meta-RaPS to solve the large instances. Table 46 presents the parameter setting of the new Meta-RaPS.

Table 46. The New Parameter Setting of Meta-RaPS V2 for Large Instances

Parameter	**Value**
Priority percentage (p)	0.60
Restriction percentage (r)	0.65
Path Relinking percentage (i)	0.10
Number of iterations (I)	10000

With this parameter setting, Meta-RaPS V2 is applied to solve all large size 0-1 MKP instances due to its fast computation. Its solution summary is presented in Table 47. The new algorithm could find optimum values for 26, 15 and 3 instances of the 100 items and 5, 10 and 30 knapsacks, respectively, and the average optimum solution for 30

instances is 14.7. The average deviation from optimum/best solution is 0.211% in average of 0.2 minutes and 2435 iterations. The overall average deviations percentage for CSs is 0.35. Meta-RaPS V2 could find the average optimum/best solution 3.7 times in 10 replications. The results for the first three sets are parallel with the overall results for all set of instances. The average deviations percentage from optimum/best solution reached by Meta-RaPS V2 is 0.241% in an average of 3.5 minutes and 3,600 iterations. Its average optimum solution is 6.2 in 30 instances, and the average optimum/best solution was found 1.2 times in 10 replications.

Table 47. Meta-RaPS V2 Solution for Large 0-1 MKP Instances

	Deviation%			Iteration	Time	Optimum	
Instance Set	**IMean**	**IBest**	**CMean**	**Number**	**(Min.)**	**Frequency**	**Instance**
100x5	0.025	0.007	0.122	1802	0.05	6.63	26
100x10	0.188	0.126	0.350	2692	0.13	3.93	15
100x30	0.420	0.242	0.588	2812	0.39	0.47	3
Average	**0.211**	**0.125**	**0.353**	**2435**	**0.19**	**3.67**	**14.67**
Std.Dev.	**0.199**	**0.118**	**0.233**	**552**	**0.18**	**3.09**	**11.50**
250x5	0.081	0.036	0.171	4132	0.75	1.17	7
250x10	0.201	0.119	0.361	4275	1.34	0.17	3
250x30	0.487	0.354	0.764	4034	3.07	0.00	0
500x5	0.115	0.077	0.214	4403	3.93	0.07	2
500x10	0.194	0.132	0.329	4502	6.58	0.00	0
500x30	0.458	0.330	0.741	3818	15.51	0.00	0
Overall Average	**0.241**	**0.158**	**0.404**	**3608**	**3.53**	**1.24**	**6.22**
Overall Std.Dev.	**0.171**	**0.124**	**0.239**	**943**	**4.99**	**2.34**	**8.83**

As seen in the first three rows in Table 47, which represent the solutions of the instance sets 100 items and 5, 10 and 30 knapsacks, Meta-RaPS V2 without improvement phase but with more sophisticated PR approach can produce promising results with much lower computational time, comparing with Meta-RaPS PR with improvement phase and a basic PR approach presented in Table 40.

CHAPTER 10

CONCLUSIONS

The proposed Meta-RaPS versions presented in the previous chapters attempt to incorporate memory and learning into Meta-RaPS from different perspectives. While Estimation of Distribution Algorithms (EDA) is a statistical learning-based approach, Q Learning takes a machine learning approach. The common part shared by both approaches is requiring a learning set that needs to be trained prior to solving the problem by one of the algorithms. On the other hand, Path Relinking (PR) makes the algorithm learn the "good" attributes by memorizing best solutions, and following them to reach better solutions. This type of learning does not need any learning sets, and can be defined as a post-optimization method. The last perspective of incorporating memory and learning is about tuning parameters, which is vital for a metaheuristic's performance. Thus, the last proposed version of Meta-RaPS has the ability of tuning parameters adaptively.

10.1 Comparison of Meta-RaPS Versions for Small/Medium 0-1 MKP Instances

These proposed algorithms showed different performance levels when applied to both small/medium and large size 0-1 Multidimensional Knapsack problems. For small/medium size instances, the algorithms were tested with and without using a greedy rule, i.e. Dynamic Greedy Rule (DGR). The purpose of such an effort was to understand how the algorithms behave independently when there is no help from a greedy rule.

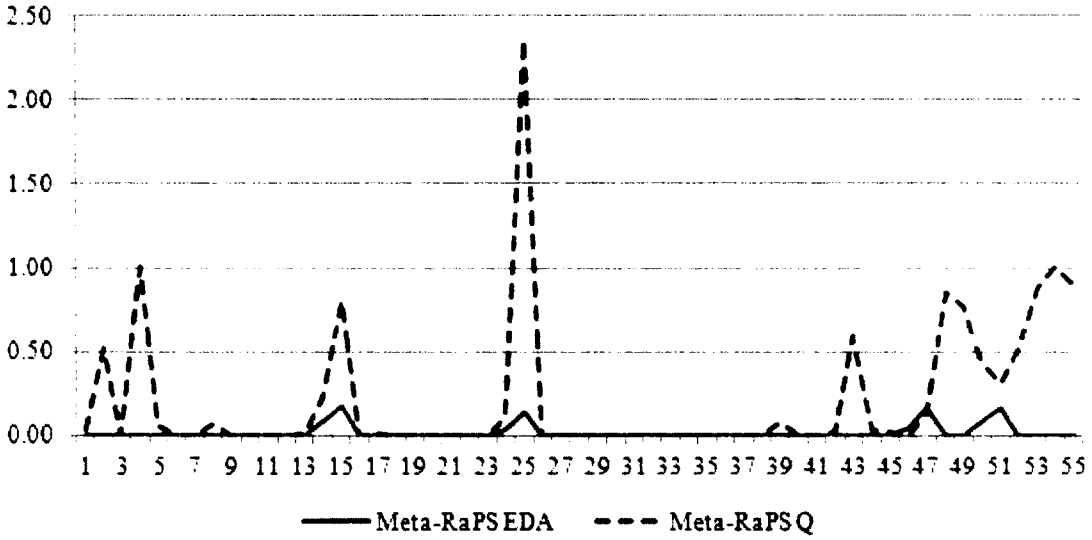

Figure 22a. Trends for Average Deviations% of Small/Medium 0-1 MKP Instances for Meta-RaPS EDA and Meta-RaPS Q without Using DGR

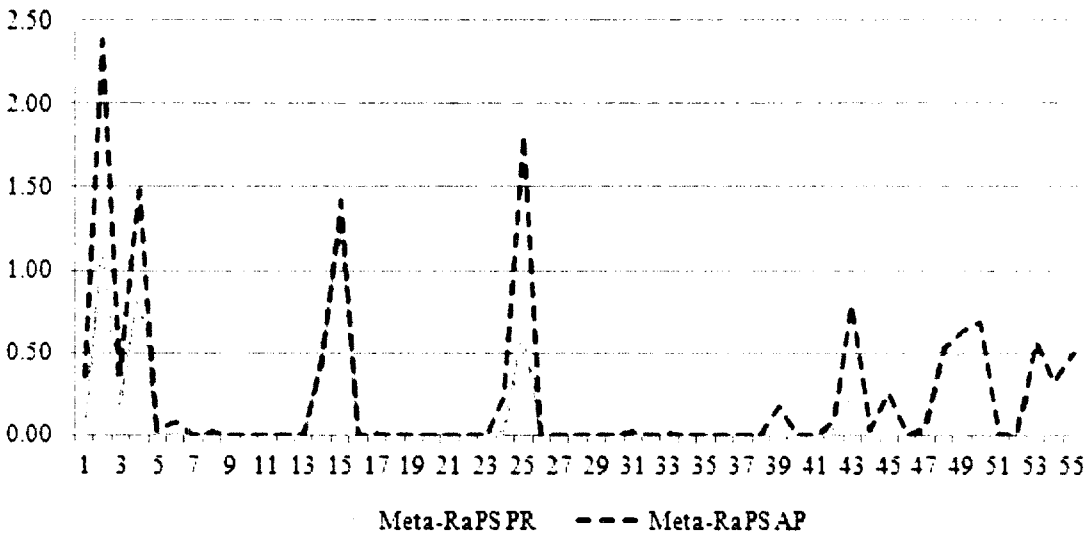

Figure 22b. Trends for Average Deviations% of Small/Medium 0-1 MKP Instances for Meta-RaPS PR, Meta-RaPS AP and Meta-RaPS V2 without Using DGR

Even in this case, Meta-RaPS EDA could create good results with the average percentage deviation of 0.016 and obtaining optimal solutions for 54 instances out of 55 instances. This phenomenon can also be observed from the trends for average deviations% of small/medium 0-1 MKP instances shown in the Figures 22a and b. Using DGR in the different versions of the proposed algorithms showed better performance than those without DGR (see Table 49), and therefore, they were accepted as the true versions of the proposed Meta-RaPS algorithms. Table 49 presents the comparison of these Meta-RaPS versions with using DGR for small/medium 0-1 MKP instances. In terms of the average deviation percentage, Meta-RaPS EDA and PR gave the best solutions where both algorithms could solve all instances. Meta-RaPS Q gave the worst outcomes among all five versions with the highest deviation percentage and number of iterations; it could find the optimum solutions for only 52 instances. Meta-RaPS AP could produce rather good results in all aspects, and reach the optimum solutions for 54 instances. Although Meta-RaPS V2 does not have the best average percentage deviation, it could find the optimum solutions for 55 instances at least once. However, the most important advantage of this algorithm is that it is very fast, almost 400 times faster than the average solution time of all proposed algorithms.

Trends for average deviations percentages of small/medium 0-1 MKP instances for the proposed Meta-RaPS versions are shown in the Figures 23a and b. In these figures the performances of the algorithms can be tracked instance by instance, and verified with Table 48.

Table 48. Comparison of Meta-RaPS Versions for Small/Medium 0-1 MKP Instances Using DGR

	Deviation%			Iteration	Time	Optimum	
Version	**IMean**	**IBest**	**CMean**	**Number**	**(Sec.)**	**Frequency**	**Instance**
Meta-RaPS EDA	0.001	0.000	0.107	421	120.09	9.84	55
Meta-RaPS Q	0.045	0.003	0.596	2074	209.19	7.96	52
Meta-RaPS PR	0.001	0.000	1.282	480	47.93	9.76	55
Meta-RaPS AP	0.012	0.002	0.107	506	257.34	9.55	54
Meta-RaPS V2	0.010	0.000	0.938	416	0.32	9.86	55
Average	**0.014**	**0.001**	**0.606**	**779**	**126.97**	**9.39**	**54.20**
Std.Dev.	**0.018**	**0.001**	**0.516**	**725**	**107.34**	**0.81**	**1.30**

After the comparisons of the proposed Meta-RaPS versions, Table 49 presents the comparison of these Meta-RaPS versions to other algorithms in the literature for small/medium 0-1 MKP instances. For these instances, TS methods (Glover & Kochenberger, 1996; Hanafi, et al., 1996), GA (Chu & Beasley, 1998) and Fix+cut based method (Osorio, et al., 2003) generated best results in the literature. The proposed Meta-RaPS algorithms could create considerably good results in terms of the number of optimal solutions and percentage deviations, and Meta-RaPS EDA and Meta-RaPS PR reached better results than Meta-RaPS DGR, which represents the Meta-RaPS version before memory and learning inclusion (Moraga, et al., 2005).

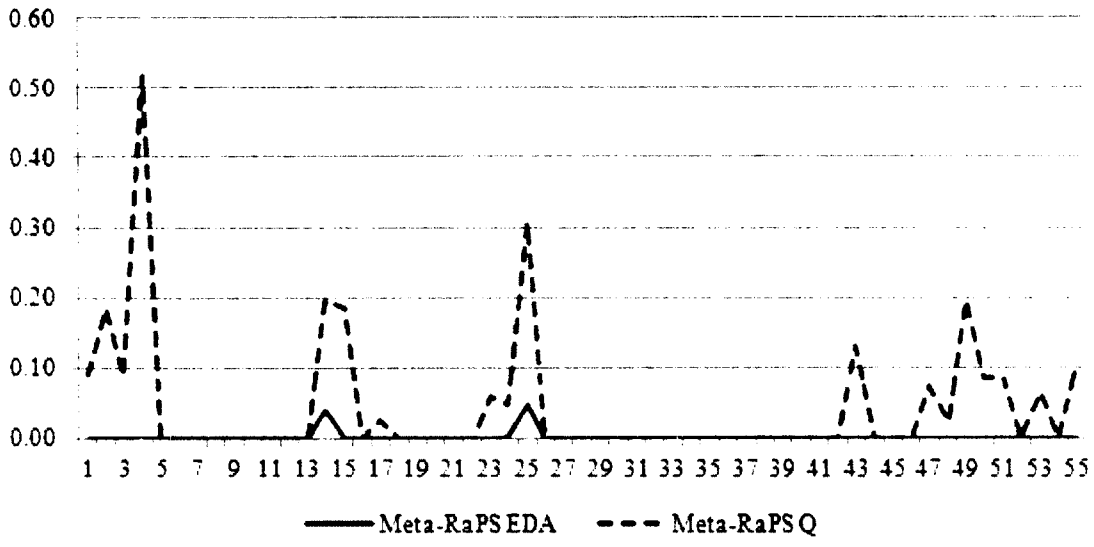

Figure 23a. Trends for Average Deviations% of Small/Medium 0-1 MKP instances for Meta-RaPS EDA and Meta-RaPS Q Using DGR

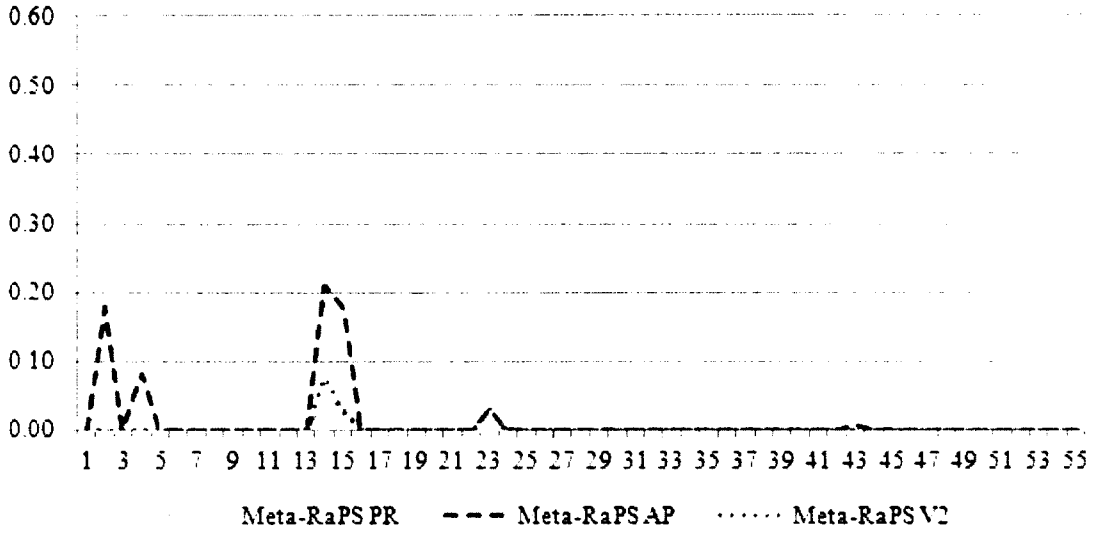

Figure 23b. Trends for Average Deviations% of Small/Medium 0-1 MKP Instances for Meta-RaPS PR, Meta-RaPS AP and Meta-RaPS V2 Using DGR

Table 49. Comparison of Meta-RaPS Versions to Other Algorithms in the Literature for Small/Medium 0-1 MKP Instances (Adapted from Moraga, et al., 2005)

Algorithm	#Optimal Solutions	Dev.%
Meta-RaPS EDA	55/55	0.001
Meta-RaPS Q	52/55	0.045
Meta-RaPS PR	55/55	0.001
Meta-RaPS AP	54/55	0.012
Meta-RaPS V2	55/55	0.010
Meta-RaPS DGR	55/56	0.003
GRASP	52/56	0.023
SMA/TA (Hanafi, et al., 1996)	39/54	0.080
AGNES (Fre´ville and Plateau, 1994 as reported by Hanafi, et al., 1996)	52/54	0.020
Tabu search REM (Dammeyer & Voss, 1993)	40/57	0.126
Tabu search STM (Dammeyer & Voss, 1993)	39/57	0.130
Tabu search L+STM (Dammeyer &Voss, 1993)	44/57	0.101
Tabu search (Glover & Kochenberger, 1996)	57/57	0.000
Tabu search (Lokketangen & Glover, 1998)	37/54	0.003
Tabu search IFTS/HFE (Hanafi, et al., 1996)	54/54	0.000
Genetic algorithm (Chu & Beasley, 1998)	55/55	0.000
Fix+cut based method (Osorio, et al., 2003)	55/55	0.000
Simulated annealing DETEXC (Drexel, 1988)	7/57	1.739
Simulated annealing PROEXC (Drexel, 1988)	23/57	0.239
Simulated annealing (Drexel, 1988 as implemented by Dammeyer & Voss, 1993)	31/57	0.328

10.2 Comparison of Meta-RaPS Versions for Large 0-1 MKP Instances

Altough the analysis for the small/medium size instances gave an idea about the performance of the proposed algorithms, it is not enough to make a conclusion about their qualities since the differences in the average percentage deviations are so small and

the instances solved are relatively easy. Therefore the proposed algorithms had to be tested on large instances. Table 50 summarizes the comparison of Meta-RaPS versions for large 0-1 MKP instances. For large size instances, all algorithms except Meta-RaPS Q presented close performance, and the solution time advantage of Meta-RaPS V2 was still remarkable with being around 200 times faster than the overall average solution time. The details of this comparison can be found in Table 51. In Section 4.3 it was stated that the large instances were created by accepting the tightness ratios of 0.25, 0.50 and 0.75 for each group of 10 instances in each set. Based on these tightness ratios, the trends of average percentage deviations of large instances for Meta-RaPS algorithms are presented in Figures 24a and b. As Pirkul (1987) pointed that, for the lower tightness ratios of the instances, the hardness of the instances are increased and as a result, the algorithms produced solutions with higher percentage deviations. In the case of higher tightness ratios, all algorithms produced better results. This phenomenon is clearer in the case of Meta-RaPS Q.

Table 50. Comparison of Meta-RaPS Versions for Large 0-1 MKP Instances

	Deviation%			Iteration	Time	Optimum	
Version	**IMean**	**IBest**	**CMean**	**Number**	**(Min.)**	**Frequency**	**Instance**
Meta-RaPS EDA	0.142	0.084	0.540	1872	50.37	3.57	15.67
Meta-RaPS Q	0.452	0.273	1.171	4176	87.39	2.36	12.67
Meta-RaPS PR	0.146	0.061	0.640	2988	20.88	4.87	21.33
Meta-RaPS AP	0.115	0.046	0.452	2081	43.59	5.59	22.00
Meta-RaPS V2	0.211	0.125	0.353	2435	0.19	3.67	14.67
Average	**0.213**	**0.118**	**0.631**	**2710**	**40.48**	**4.01**	**17.27**
Std.Dev.	**0.138**	**0.092**	**0.320**	**922**	**32.85**	**1.25**	**4.16**

Table 51. Detailed Comparison of Meta-RaPS Versions for Large 0-1 MKP Instances

Instance Set	Deviation%			Iteration	Time	Optimum	
Version	IMean	IBest	CMean	Number	(Min.)	Frequency	Instance
100x5							
Meta-RaPS EDA	0.045	0.026	0.296	1389	16.19	5.17	20
Meta-RaPS Q	0.143	0.058	0.653	3440	19.67	4.97	22
Meta-RaPS PR	0.031	0.008	0.353	2321	14.44	7.27	28
Meta-RaPS AP	0.019	0.005	0.081	1329	25.34	7.90	28
Meta-RaPS V2	0.025	0.007	0.122	1802	0.05	6.63	26
Average	**0.053**	**0.021**	**0.301**	**2056**	**15.14**	**6.39**	**24.80**
Std.Dev.	**0.051**	**0.022**	**0.228**	**869**	**9.41**	**1.23**	**3.63**
100x10							
Meta-RaPS EDA	0.136	0.078	0.519	1869	33.79	3.90	17
Meta-RaPS Q	0.493	0.281	1.261	4454	39.35	1.50	10
Meta-RaPS PR	0.155	0.080	0.650	2819	27.32	5.17	22
Meta-RaPS AP	0.098	0.050	0.501	2215	20.72	6.17	23
Meta-RaPS V2	0.188	0.126	0.350	2692	0.13	3.93	15
Average	**0.214**	**0.123**	**0.656**	**2810**	**24.26**	**4.14**	**17.40**
Std.Dev.	**0.159**	**0.092**	**0.354**	**995**	**15.19**	**1.76**	**5.32**
100x30							
Meta-RaPS EDA	0.246	0.147	0.804	2357	101.13	1.63	10
Meta-RaPS Q	0.721	0.479	1.599	4634	98.28	0.60	6
Meta-RaPS PR	0.252	0.095	0.917	3824	86. 17	2.17	14
Meta-RaPS AP	0.227	0.083	0.775	2698	84.70	2.67	15
Meta-RaPS V2	0.420	0.242	0.588	2812	0.39	0.47	3
Average	**0.373**	**0.209**	**0.937**	**3265**	**71.13**	**1.51**	**9.60**
Std.Dev.	**0.209**	**0.163**	**0.389**	**941**	**47.70**	**0.96**	**5.13**
Overall Average	**0.213**	**0.118**	**0.631**	**2710**	**34.39**	**4.01**	**17.27**
Overall Std.Dev.	**0.197**	**0.129**	**0.408**	**1009**	**34.94**	**2.42**	**7.79**

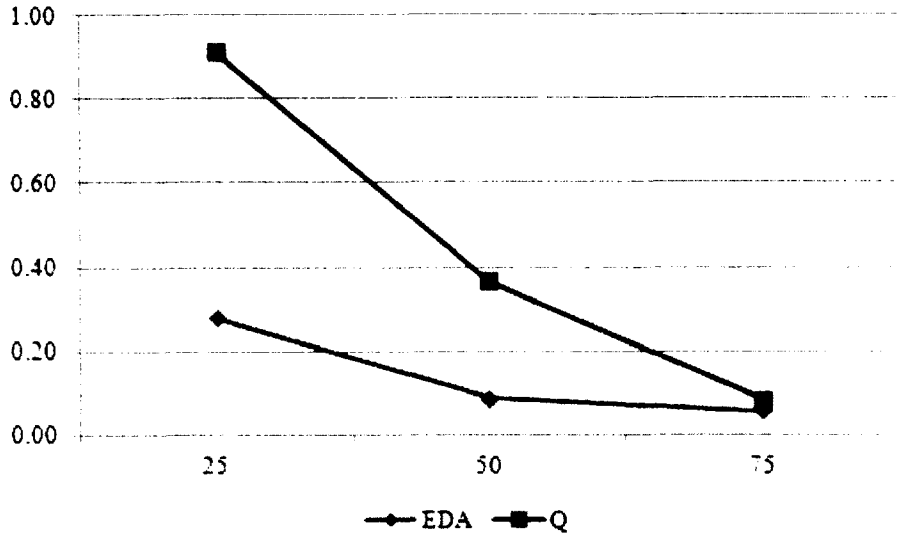

Figure 24a. Trends of Average Deviations% of Large Instances for Meta-RaPS EDA and Meta-RaPS Q Based on Instance Tightness Ratios

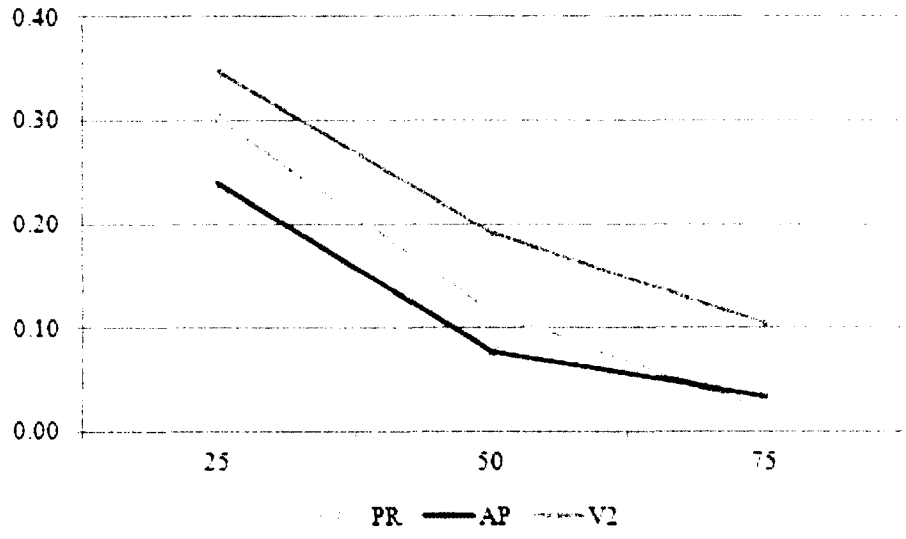

Figure 24b. Trends of Average Deviations% of Large Instances for Meta-RaPS PR, Meta-RaPS AP and Meta-RaPS V2 Based on Instance Tightness Ratios

Along with the comparisons of the proposed algoritms presented until now, they can also be compared in a more systematic way. In the literature, five major performance criteria are described against which good metaheuristics can be measured: accuracy, speed, simplicity, flexibility, and consistency (Cordeau, et al., 2002; Laporte, et al., 2000; Wassan, 2006). To elaborate, a proposed algorithm can be measured according to these performance criteria as follows:

1. Accuracy: How accurate the results are when applied to various problems. This reflects the quality of the algorithm's results.
2. Speed: CPU time spent by the algorithms to solve instances,
3. Simplicity: The convenience of using the algorithm with the problems at hand,
4. Flexibility: The convenience of modifying the algorithms for different problems, and
5. Consistency: Robustness of the algorithm with different instances of different problems. In terms of computational efficiency and optimality, Ide and Yasuda (2005) define robustness as the ability of an algorithm to withstand differences in problem structures. The chance of achieving a poor solution should be a very low, and the performance should not be sensitive to the parameters (Silver, 2004).

In this study's solution reports, the values for some of these criteria have already been obtained, especially those that are quantitative in nature. Specifically, the *percentage deviation* was recorded for accuracy, *time* for speed and *standard deviation* for consistency for each proposed algorithm. Different from *percentage deviation* and *standard deviation* which are already between 0 and 1, the values of *time* for each algorithm were normalized in order to reach their scaled values in the same interval. For

the *simplicity* and *flexibility* criteria, the appropriate numerical values were given based on the average labor hours spent for using (*simplicity*) and modifying (*flexibility*) the algorithms when required to solve different type of problem. These labor hours can be easily transformed into the scaled factors between 0 and 1 by taking their ratios in overall total, as in Table 52.

Table 52. Creating Scaled Factors for Simplicity and Flexibility

Performance Criteria	**Value**	**Meta-RaPS**				
		EDA	**Q**	**PR**	**AP**	**V2**
Simplicity	Labor Hour	3	3.5	1	2	1.5
	Scaled Factor	0.60	0.70	0.20	0.40	0.30
Flexibility	Labor Hour	3.5	4	0.5	0.5	0.5
	Scaled Factor	0.70	0.80	0.10	0.10	0.10

Although there are five performance criteria to measure the performance of the metaheuristics, this does not mean that all performance criteria have the same level of significance. One criterion might be more important than others for the users, and therefore, weights can be assigned to each based on users' experience. The values of the *percentage deviation* for accuracy, *time* for speed and *standard deviation* for consistency were calculated from the runs. It is assumed in this research that the weights of *simplicity* and *flexibility* as 15% in total.

In Table 53, the comparison of Meta-RaPS versions in terms of performance criteria is presented. For the performance criteria, the lower values are always better. The total weighted values for each proposed algorithm were calculated by multiplying the

values for the performance criteria by their weights and then summing up these weighted values. As seen in Table 53, the minimum and maximum total weighted values belong to Meta-RaPS PR and Meta-RaPS Q algorithms, respectively. Therefore, it can be argued that Meta-RaPS PR has the best performance and Meta-RaPS Q has the worst performance among the all proposed versions.

Table 53. Comparison of Meta-RaPS Versions in Terms of Performance Criteria

Performance Criteria	**Definition**	**Weight**	**Meta-RaPS**				
			EDA	**Q**	**PR**	**AP**	**V2**
Accuracy	Deviation%	0.50	0.15	0.45	0.15	0.12	0.21
Speed	Time	0.20	0.25	0.43	0.10	0.22	0.00
Simplicity	Scaled Factor (0-1)	0.05	0.60	0.70	0.20	0.40	0.30
Flexibility	Scale Factor (0-1)	0.10	0.70	0.80	0.10	0.10	0.10
Consistency	Standard Deviation	0.15	0.10	0.29	0.11	0.11	0.20
Total Weighted Values		**1.00**	**0.24**	**0.47**	**0.13**	**0.15**	**0.16**

To observe the effects of the given weights on the total weighted values for each Meta-RaPS version, the sensitivities of the total weighted values for the first two largest weights (deviation% and time), are analyzed in Figures 25 and 26. In this sensititity analysis, while these weights are increasing/decreasing, their amount of change is equally subtracted from/added to the weights of the other performance criteria. As can be seen in these figures, selection of the best algorithm, Meta-RaPS PR, does not changed for large intervals of the weights of deviation% and time. This means that the weights are fairly robust in obtaining the total weighted values.

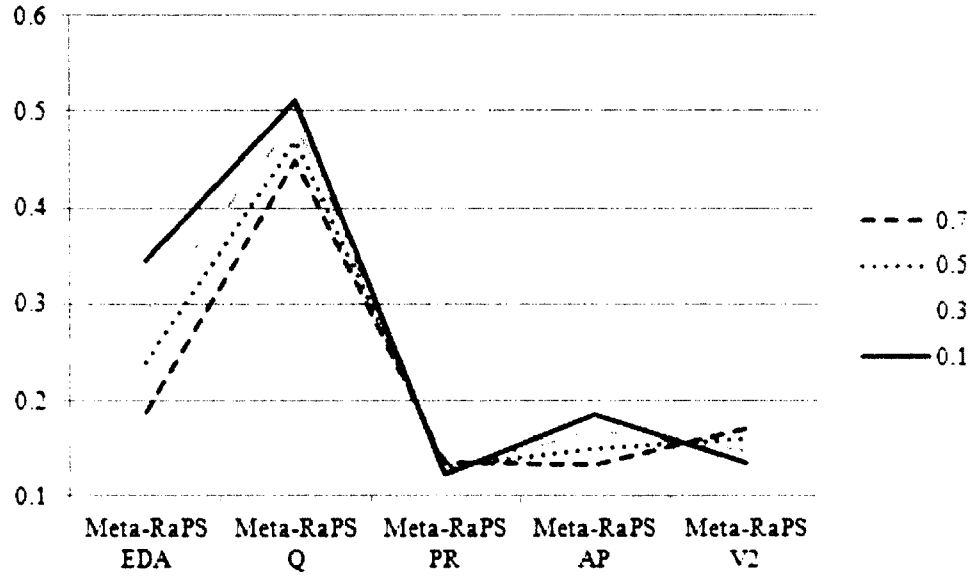

Figure 25. Sensitivity of Total Weighted Values for Different Weights of *Percentage Deviation*

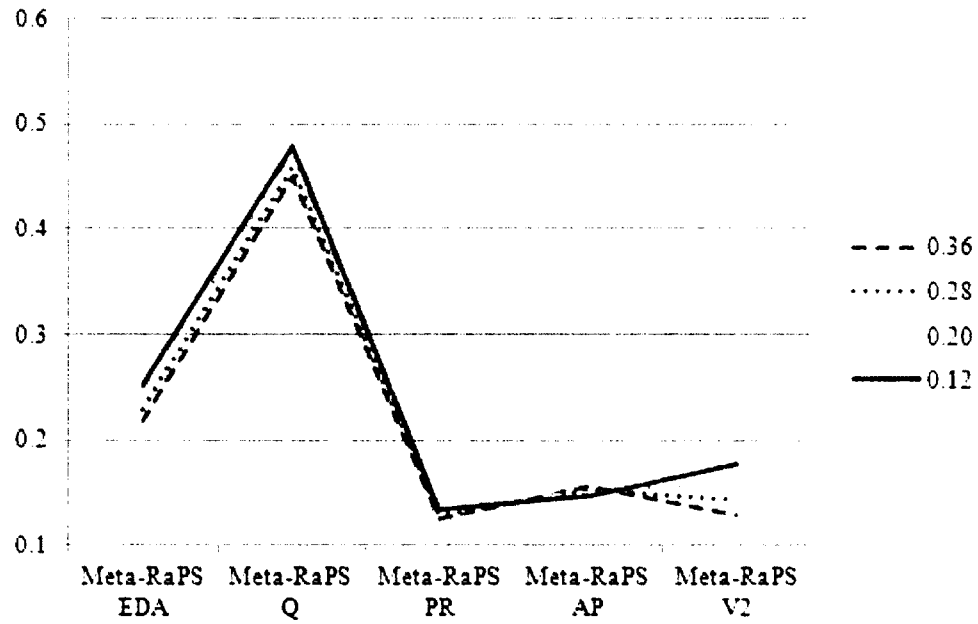

Figure 26. Sensitivity of Total Weighted Values for Different Weights of *Time*

The comparison of the proposed Meta-RaPS algorithms to other algorithms in the literature for large 0-1 MKP instances is presented in Table 54 in terms of percentage deviation. Besides all the versions of Meta-RaPS, the Genetic Algorithm approaches of Chu and Beasley (1998) and Haul and Voss (1997) denoted by GA-CB and GA-HV, respectively were included in the comparison. Furthermore, the algorithms of Magazine and Oguz (1984), Pirkul (1987), and Volgenant and Zoon (1990) denoted by MKP-P, MKP-MO, MKP-VZ, respectively, as well as the Approximate Dynamic Programming (ADP) method of Bertsimas and Demir (2002) were also included in the comparison. In some cells of Table 54, there are no entries because Meta-RaPS EDA, Q, PR and AP were applied to the sets of 100 items with 5, 10 and 30 knapsacks, and in ADP approach, only the results for the sets of 250 and 500 items with 30 knapsacks were reported.

In the first three sets (100 items and 5, 10 and 30 knapsacks) all proposed Meta-RaPS algorithms produced better results than the other algorithms in the literature. After redesigning the Meta-RaPS by removing the improvement phase (local search) and adding a more sophisticated Path Relinking approach, Meta-RaPS V2 was used to solve all of the large instances. Although the approach of Bertsimas and Demir's (2002) shares the best overall percentage deviation with Meta-RaPS V2 among the other algorithms, it can be observed that Meta-RaPS V2 outperformed ADP if their results for the instance sets of 250x30 and 500x30 are compared. For the large size 0-1 MKP instances, Moraga, et al. (2005) reported that Meta-RaPS DGR run times ranged from 7 to 35 minutes per problem, and average solution times for Chu and Beasley's (1998) Genetic Algorithm ranged from 6 to 65 minutes on a Silicon Graphics Indigo workstation.

Table 54. Comparison of Meta-RaPS Versions to Other Algorithms for Large 0-1 MKP Instances (Adapted from Moraga, et al., 2005)

Instance Set	Meta-RaPS											
	EDA	**Q**	**PR**	**AP**	**MR2**	**DGR**	**GA-CB**	**GA-HV**	**MKP-P**	**MK-MO**	**MKP-VZ**	**ADP**
100x5	0.05	0.14	0.03	0.02	0.03	0.60	0.59	0.72	0.95	8.49	7.63	N/A
100x10	0.14	0.49	0.16	0.10	0.19	1.17	0.94	1.26	2.12	10.79	10.65	N/A
100x30	0.25	0.72	0.25	0.23	0.42	2.23	1.69	2.14	4.85	11.93	11.11	N/A
Average	**0.15**	**0.45**	**0.15**	**0.12**	**0.21**	**1.33**	**1.07**	**1.37**	**2.64**	**10.40**	**9.80**	**N/A**
Std.Dev.	**0.10**	**0.29**	**0.11**	**0.11**	**0.20**	**0.83**	**0.56**	**0.72**	**2.00**	**1.75**	**1.89**	**N/A**
250x5	N/A	N/A	N/A	N/A	0.08	0.17	0.14	0.36	0.31	5.14	4.61	N/A
250x10	N/A	N/A	N/A	N/A	0.20	0.45	0.30	0.74	0.66	7.66	6.74	N/A
250x30	N/A	N/A	N/A	N/A	0.49	1.38	0.68	1.36	2.02	8.89	7.81	0.97
500x5	N/A	N/A	N/A	N/A	0.12	0.09	0.05	0.34	0.12	3.40	3.02	N/A
500x10	N/A	N/A	N/A	N/A	0.19	0.20	0.14	0.64	0.29	6.05	4.99	N/A
500x30	N/A	N/A	N/A	N/A	0.46	0.82	0.35	1.20	1.03	6.89	6.28	0.52
Overall Average	**N/A**	**N/A**	**N/A**	**N/A**	**0.24**	**0.79**	**0.54**	**0.97**	**1.37**	**7.69**	**6.98**	**0.24**
Overall Std.Dev.	**N/A**	**N/A**	**N/A**	**N/A**	**0.17**	**0.70**	**0.52**	**0.58**	**1.49**	**2.69**	**2.68**	**0.17**

The algorithm of Haul and Voss (1997) takes a very long time to solve large instances, in some cases more than four hours on an Intel Pentium 100 MHz PC. Bertsimas and Demir's (2002) reported an average solution time of 87 seconds on a Dell Precision 410 machine. Other than the approach of Bertsimas and Demir's (2002), Meta-RaPS V2 algorithm outperformed all other algorithms in terms of CPU with an average solution time of 3.53 minutes (211 seconds).

10.3 Statistical Comparison of Meta-RaPS Versions

To begin the statistical comparison of the different Meta-RaPS versions, first, the method of the one-way ANOVA is applied to understand if there is significant difference among the proposed algorithms in terms of the means of Percentage Deviations and Time for Meta-RaPS Versions. For the one-way ANOVA, the following hypotheses are constructed:

H_0: Means of percentage deviations/time for Meta-RaPS versions are equal.

H_1: At least one of the means of percentage deviations/time for Meta-RaPS versions is different.

First, the means of percentage deviations of the Meta-RaPS versions will be evaluated for the set of 100 Items and 5 Knapsacks. One of the assumptions of the one-way ANOVA is that the variances of the groups are similar. The Test of Homogeneity of Variances in Table 54 shows the result of Levene's Test of Homogeneity of Variance, which tests for similiar variances. If the significance value is greater than 0.05 then there is an homogeneity of variances. Levene's *F* Statistic has a significance value of 0.00 and, thus, the assumption of homogeneity of variance is not met. When there is a violation of

the assumption of homogeneity of variances, the significant difference between the groups could still be determined by the Welch test in the Robust Tests of Equality of Means (Table 55). Since the significance value of Welch test is 0.01, which is less than 0.05, then we can say that there is statistically significant difference between groups.

Table 55. ANOVA of *Percentage Deviation* for the Set of 100 Items and 5 Knapsacks

Test of Homogeneity of Variances

Deviation

Levene Statistic	df1	df2	Sig.
26.370	4	145	.000

ANOVA

Deviation

	Sum of Squares	df	Mean Square	F	Sig.
Between Groups	.315	4	.079	8.446	.000
Within Groups	1.353	145	.009		
Total	1.668	149			

Robust Tests of Equality of Means

Deviation

	Statistic[a]	df1	df2	Sig.
Welch	3.627	4	70.958	.010

a. Asymptotically F distributed.

Because the null hypothesis (*H_0: Means of Percentage Deviations for Meta-RaPS Versions Are Equal*) is rejected via the one-way ANOVA, the Tukey's multiple comparison test is conducted as the next step. In this analysis, the Meta-RaPS EDA, Q, PR, AP and V2 versions are coded as numbers from 1 to 5, respectively. According to results in terms of percentage deviations shown in Table 56, only Meta-RaPS Q is staisically different from the other versions, and there are no statistically significant differences among Meta-RaPS EDA, PR, AP and V2 versions. The obtained results are

not different for other set of instances (100 items and 10, 30 knapsacks) solved by the proposed algorithms, as shown in Tables 57 and 58.

Table 56. Tukey's Test of *Percentage Deviation* for the Set 100 Items and 5 Knapsacks

Deviation
Tukey HSD

(I) Algorithm	(J) Algorithm	Mean Difference (I-J)	Std. Error	Sig.	95% Confidence Interval	
					Lower Bound	Upper Bound
1	2	-.09771*	.02494	.001	-.1666	-.0288
	3	.01374	.02494	.982	-.0552	.0826
	4	.02600	.02494	.835	-.0429	.0949
	5	.01967	.02494	.934	-.0492	.0886
2	1	.09771*	.02494	.001	.0288	.1666
	3	.11144*	.02494	.000	.0426	.1803
	4	.12370*	.02494	.000	.0548	.1926
	5	.11738*	.02494	.000	.0485	.1863
3	1	-.01374	.02494	.982	-.0826	.0552
	2	-.11144*	.02494	.000	-.1803	-.0426
	4	.01226	.02494	.988	-.0566	.0811
	5	.00593	.02494	.999	-.0630	.0748
4	1	-.02600	.02494	.835	-.0949	.0429
	2	-.12370*	.02494	.000	-.1926	-.0548
	3	-.01226	.02494	.988	-.0811	.0566
	5	-.00633	.02494	.999	-.0752	.0626
5	1	-.01967	.02494	.934	-.0886	.0492
	2	-.11738*	.02494	.000	-.1863	-.0485
	3	-.00593	.02494	.999	-.0748	.0630
	4	.00633	.02494	.999	-.0626	.0752

*. The mean difference is significant at the 0.05 level.

These outcomes were expected because our main focus was on integrating memory and learning mechanisms into Meta-RaPS, and as specified in 4.6, the only difference among all these proposed algorithms is how to produce the priorities for Meta-RaPS to use in the solution process. In the original Meta-RaPS version, the algorithm needs to apply some greedy rules to select the next item to add to the partial solution.

Table 57. Tukey's Test of *Percentage Deviation* for the Set 100 Items and 10 Knapsacks

Deviation
Tukey HSD

(I) Algorithm	(J) Algorithm	Mean Difference (I-J)	Std. Error	Sig.	95% Confidence Interval	
					Lower Bound	Upper Bound
1	2	-.35701*	.07305	.000	-.5588	-.1552
	3	-.01867	.07305	.999	-.2205	.1831
	4	.03794	.07305	.985	-.1639	.2397
	5	-.05220	.07305	.953	-.2540	.1496
2	1	.35701*	.07305	.000	.1552	.5588
	3	.33834*	.07305	.000	.1365	.5401
	4	.39495*	.07305	.000	.1931	.5967
	5	.30481*	.07305	.000	.1030	.5066
3	1	.01867	.07305	.999	-.1831	.2205
	2	-.33834*	.07305	.000	-.5401	-.1365
	4	.05660	.07305	.937	-.1452	.2584
	5	-.03353	.07305	.991	-.2353	.1683
4	1	-.03794	.07305	.985	-.2397	.1639
	2	-.39495*	.07305	.000	-.5967	-.1931
	3	-.05660	.07305	.937	-.2584	.1452
	5	-.09013	.07305	.732	-.2919	.1117
5	1	.05220	.07305	.953	-.1496	.2540
	2	-.30481*	.07305	.000	-.5066	-.1030
	3	.03353	.07305	.991	-.1683	.2353
	4	.09013	.07305	.732	-.1117	.2919

*. The mean difference is significant at the 0.05 level.

Table 58. Tukey's Test for *Percentage Deviation* of the Set 100 Items and 30 Knapsacks

Deviation
Tukey HSD

(I) Algorithm	(J) Algorithm	Mean Difference (I-J)	Std. Error	Sig.	95% Confidence Interval	
					Lower Bound	Upper Bound
1	2	-.47429*	.08847	.000	-.7187	-.2299
	3	-.00518	.08847	1.000	-.2496	.2392
	4	.01960	.08847	.999	-.2248	.2640
	5	-.17399	.08847	.288	-.4184	.0704
2	1	.47429*	.08847	.000	.2299	.7187
	3	.46911*	.08847	.000	.2247	.7135
	4	.49389*	.08847	.000	.2495	.7383
	5	.30030*	.08847	.008	.0559	.5447
3	1	.00518	.08847	1.000	-.2392	.2496
	2	-.46911*	.08847	.000	-.7135	-.2247
	4	.02478	.08847	.999	-.2196	.2692
	5	-.16881	.08847	.318	-.4132	.0756
4	1	-.01960	.08847	.999	-.2640	.2248
	2	-.49389*	.08847	.000	-.7383	-.2495
	3	-.02478	.08847	.999	-.2692	.2196
	5	-.19359	.08847	.190	-.4380	.0508
5	1	.17399	.08847	.288	-.0704	.4184
	2	-.30030*	.08847	.008	-.5447	-.0559
	3	.16881	.08847	.318	-.0756	.4132
	4	.19359	.08847	.190	-.0508	.4380

*. The mean difference is significant at the 0.05 level.

However, the proposed Meta-RaPS versions benefits from the intelligent approaches to understand the structure of the problem and make intelligent decisions in reaching high quality solutions. The main structure of the Meta-RaPS was kept the same while designing the proposed versions of Meta-RaPS.

Even though the difference among the proposed algorithms is small, the difference created by these algorithms in the solution quality is striking. When compared to other approaches in the literature, all the Meta-RaPS versions could generate very promising results, except the Meta-RaPS Q version. This is the reason why there are not statistically significant differences among the Meta-RaPS EDA, PR, AP and V2 versions, and only Meta-RaPS Q is different from the other versions.

The "time" factor will be the next focus to analyze the Meta-RaPS versions for the set of 100 Items and 5 Knapsacks. Again the Levene's Statistics in Test of Homogeneity of Variances shown in Table 59 is 0.00, which is less than 0.05, and the assumption of homogeneity of variance is not met. Thus, the Welch test in the Robust Tests of Equality of Means can be used to make a judgment. The significance value of Welch test is 0.00, less than 0.05, and therefore it can be concluded that there are statistically significant differences between the groups. The null hypothesis (*H_0: Means of Time for Meta-RaPS Versions Are Equal*) is rejected, which means that at least one of the algorithms is different.

To reveal the different Meta-RaPS versions in terms of time, the Tukey's multiple comparison test was conducted again for the sets having the instances 100 items and 5, 10, 30 knapsacks. As presented in Tables 60-62, for the first set, Meta-RaPS EDA and PR are different from both Meta-RaPS AP and V2; Meta-RaPS Q is different from the

Meta-RaPS V2; Meta-RaPS AP is different from the Meta-RaPS EDA, AP and V2; and Meta-RaPS V2 is different from all other Meta-RaPS versions.

Table 59. ANOVA of *Time* for the Set of 100 Items and 5 Knapsacks

Test of Homogeneity of Variances

Time

Levene Statistic	df1	df2	Sig.
35.375	4	145	.000

ANOVA

Time

	Sum of Squares	df	Mean Square	F	Sig.
Between Groups	3.820E7	4	9550534.633	20.501	.000
Within Groups	6.755E7	145	465845.855		
Total	1.057E8	149			

Robust Tests of Equality of Means

Time

	Statistic[a]	df1	df2	Sig.
Welch	114.786	4	58.002	.000

a. Asymptotically F distributed.

For the second set, Meta-RaPS EDA and Q are different from the versions of Meta-RaPS AP and V2; and Meta-RaPS V2 is different from all of the other Meta-RaPS versions. For the third and largest set, only Meta-RaPS V2 is different from all the other Meta-RaPS versions.

Table 60. Tukey's Test for *Time* for the Set of 100 Items and 5 Knapsacks

Time
Tukey HSD

(I) Algorithm	(J) Algorithm	Mean Difference (I-J)	Std. Error	Sig.	95% Confidence Interval	
					Lower Bound	Upper Bound
1	2	-208.99806	176.22823	.759	-695.8120	277.8158
	3	105.14770	176.22823	.975	-381.6662	591.9616
	4	-548.84768*	176.22823	.019	-1035.6616	-62.0338
	5	968.16225*	176.22823	.000	481.3483	1454.9762
2	1	208.99806	176.22823	.759	-277.8158	695.8120
	3	314.14576	176.22823	.388	-172.6681	800.9597
	4	-339.84962	176.22823	.307	-826.6635	146.9643
	5	1177.16031*	176.22823	.000	690.3464	1663.9742
3	1	-105.14770	176.22823	.975	-591.9616	381.6662
	2	-314.14576	176.22823	.388	-800.9597	172.6681
	4	-653.99538*	176.22823	.003	-1140.8093	-167.1815
	5	863.01455*	176.22823	.000	376.2006	1349.8285
4	1	548.84768*	176.22823	.019	62.0338	1035.6616
	2	339.84962	176.22823	.307	-146.9643	826.6635
	3	653.99538*	176.22823	.003	167.1815	1140.8093
	5	1517.00993*	176.22823	.000	1030.1960	2003.8238
5	1	-968.16225*	176.22823	.000	-1454.9762	-481.3483
	2	-1177.16031*	176.22823	.000	-1663.9742	-690.3464
	3	-863.01455*	176.22823	.000	-1349.8285	-376.2006
	4	-1517.00993*	176.22823	.000	-2003.8238	-1030.1960

*. The mean difference is significant at the 0.05 level.

In the aspect of solution time, the statistical analysis shows that there are statistically significant differences between the Meta-RaPS versions. On the other hand, the difference between the proposed algorithms in terms of percentage deviation was small. These two criteria are not independent of each other and we need to consider both of the percentage deviation and time simultaneously where the proposed algorithms require different times to reach these percentage deviations, which are not much different except for Meta-RaPS Q. This phenomenon shows that the proposed Meta-RaPS versions employ different mechanisms in reaching these high quality solutions, and thus, it can be propounded that they are different algorithms.

Table 61. Tukey's Test for *Time* for the Set of 100 Items and 10 Knapsacks

Time
Tukey HSD

(I) Algorithm	(J) Algorithm	Mean Difference (I-J)	Std. Error	Sig.	95% Confidence Interval	
					Lower Bound	Upper Bound
1	2	-333.38614	273.02452	.739	-1087.5907	420.8184
	3	388.15719	273.02452	.615	-366.0474	1142.3617
	4	784.01010*	273.02452	.037	29.8056	1538.2147
	5	2019.88132*	273.02452	.000	1265.6768	2774.0859
2	1	333.38614	273.02452	.739	-420.8184	1087.5907
	3	721.54333	273.02452	.068	-32.6612	1475.7479
	4	1117.39624*	273.02452	.001	363.1917	1871.6008
	5	2353.26746*	273.02452	.000	1599.0629	3107.4720
3	1	-388.15719	273.02452	.615	-1142.3617	366.0474
	2	-721.54333	273.02452	.068	-1475.7479	32.6612
	4	395.85291	273.02452	.597	-358.3516	1150.0575
	5	1631.72413*	273.02452	.000	877.5196	2385.9287
4	1	-784.01010*	273.02452	.037	-1538.2147	-29.8056
	2	-1117.39624*	273.02452	.001	-1871.6008	-363.1917
	3	-395.85291	273.02452	.597	-1150.0575	358.3516
	5	1235.87122*	273.02452	.000	481.6667	1990.0758
5	1	-2019.88132*	273.02452	.000	-2774.0859	-1265.6768
	2	-2353.26746*	273.02452	.000	-3107.4720	-1599.0629
	3	-1631.72413*	273.02452	.000	-2385.9287	-877.5196
	4	-1235.87122*	273.02452	.000	-1990.0758	-481.6667

*. The mean difference is significant at the 0.05 level.

Table 62. Tukey's Test for *Time* for the Set of 100 Items and 30 Knapsacks

Time
Tukey HSD

(I) Algorithm	(J) Algorithm	Mean Difference (I-J)	Std. Error	Sig.	95% Confidence Interval	
					Lower Bound	Upper Bound
1	2	170.82767	583.36805	.998	-1440.6716	1782.3270
	3	897.68455	583.36805	.539	-713.8148	2509.1839
	4	985.72061	583.36805	.444	-625.7787	2597.2199
	5	6043.96004*	583.36805	.000	4432.4607	7655.4594
2	1	-170.82767	583.36805	.998	-1782.3270	1440.6716
	3	726.85688	583.36805	.724	-884.6424	2338.3562
	4	814.89295	583.36805	.631	-796.6064	2426.3923
	5	5873.13237*	583.36805	.000	4261.6331	7484.6317
3	1	-897.68455	583.36805	.539	-2509.1839	713.8148
	2	-726.85688	583.36805	.724	-2338.3562	884.6424
	4	88.03607	583.36805	1.000	-1523.4632	1699.5354
	5	5146.27549*	583.36805	.000	3534.7762	6757.7748
4	1	-985.72061	583.36805	.444	-2597.2199	625.7787
	2	-814.89295	583.36805	.631	-2426.3923	796.6064
	3	-88.03607	583.36805	1.000	-1699.5354	1523.4632
	5	5058.23942*	583.36805	.000	3446.7401	6669.7387
5	1	-6043.96004*	583.36805	.000	-7655.4594	-4432.4607
	2	-5873.13237*	583.36805	.000	-7484.6317	-4261.6331
	3	-5146.27549*	583.36805	.000	-6757.7748	-3534.7762
	4	-5058.23942*	583.36805	.000	-6669.7387	-3446.7401

*. The mean difference is significant at the 0.05 level.

10.4 Comments on Meta-RaPS Versions

The proposed Meta-RaPS versions created by incorporating different memory and learning mechanisms, i. e. EDA, Q Learning, PR and Adaptive Parameters, presented various performance levels in solving 0-1 MKP instances. The proposed algorithms are distinct and have different advantages and disadvantages over each other. First Meta-RaPS EDA, could produce high quality results by employing the probabilistic model for the problem after exploring the search space, dependency relationships between decision variables, and other properties of the problem landscape. However, the memory matrix that creates these advantages makes also the algorithm time consuming, in both forming and updating the memory matrix. Another issue with the memory matrix is how to reach a converged memory matrix that includes the "right" probabilities. This decision includes determining the size of memory solution set, which means adding another parameter to the Meta-RaPS parameters. Besides these factors, generating the memory matrix has more difficulties when the size of the problem increases. In such cases, the size of memory matrix should also increase requiring more computer memory and time in forming and updating the memory matrix. These obstacles might be removed by applying the appropriate methods such as function approximation methods, clustering and regression models.

Among the proposed Meta-RaPS versions, the worst outcome was generated by the version created with Q Learning. However, Meta-RaPS Q is still better than other approaches in the literature presented in Table 53. The biggest advantage of this approach is its simplicity to be applied to problems. Meta-RaPS Q shares also the disadvantages of its memory matrix where forming and updating the memory matrix of Q Learning

becomes more difficult in terms of computer memory and solution time as the problem size increases. Applying the proper methods mentioned in the case of EDA might help overcome these barriers. To make the memory matrix converge is another problem to deal with before an instance can be solved as it requires determining the size of memory solution set, which will add another parameter to Meta-RaPS's parameters. In addition to these parameters, the Q Learning algorithm has two additional parameters: learning and discount factors.

The third Meta-RaPS version was created by incorporating the PR approach, which is different from the Meta-RaPS EDA and Meta-RaPS Q in the way of applying its memory and learning ability. In the previous versions of Meta-RaPS, the memory matrix was first generated and trained so that the algorithm can successfully solve the problem by utilizing the memory matrix. However, in Meta-RaPS PR, the algorithm does not need a memory matrix, and therefore does not suffer any of these disadvantages presented earlier. Memory and learning happens by remembering the attributes of best solutions found in the solution process, and the current solution is evolved to a better solution by accepting these attributes. This approach does not need any additional parameters, and thus keeps the simplicity of Meta-RaPS.

While creating the fourth version, Meta-RaPS AP, the big impact of discovering the relationships between the parameters and the search trajectories on reaching the best solutions was the main motivation. This approach requires a parameter memory matrix as in Meta-RaPS EDA and Meta-RaPS Q, however this matrix is independent from the size of the problem, and only depends on the number of parameters focused on. This fact

makes the memory matrix converge quicker, and require less computational efforts than Meta-RaPS EDA and Meta-RaPS Q.

In Meta-RaPS PR, the very basic form of PR was applied, and the new algorithm produced very good results compared to other Meta-RaPS versions and other algorithms in the literature. Even though this approach does not have the disadvantages of the Meta-RaPS EDA and Q, it still requires high computational time with the increasing the problem size due to the improvement phase, i.e. due to local search techniques. The efficiency of PR approach encouraged us to redesign Meta-RaPS so that it can solve even the largest problems in an acceptable time frame while keeping the quality of its solutions and its design and implementation simplicity. By removing the improvement phase and adding a more sophisticated PR approach, the last version was created, Meta-RaPS V2, and reaching the goal of this dissertation.

As revealed by the experiences in creating the proposed algorithms in the previous chapters, there are convincing reasons to employ memory and learning mechanisms in metaheuristics, or intelligent algorithms, especially as the solution environment is becoming so complex that human beings can no longer understand it. Even if Meta-RaPS versions presented different performances on solving 0-1 MKP, they could all show superior performance than other approaches in the literature. This fact shows that incorporating memory and learning mechanisms into metaheuristics is a good strategy that makes the algorithms more efficient.

In the optimization area, there are such efficient metaheuristics whose power comes from their ability to memorize and learn in reaching high quality solutions for large scale problems. With the ability of learning and memorizing the search history,

these algorithms can find good initial starting point(s), and then a local method is employed to search for better solution from the initial starting point(s) (Panigrahi, et al., 2011).

In conclusion, it can be declared that implementing memory and learning mechanisms in a memoryless metaheuristic, like Meta-RaPS, can result in a significant improvement to the metaheuristic's performance. While performing essential steps of memory and learning mechanisms, i.e. generating and updating a memory matrix or reaching the best solution in PR, the new algorithm might require more computational efforts. However, there is always a cost in creating an intelligent algorithm that can memorize and learn, which can be thought of as a small cost to pay to reach high quality solutions.

Intelligence emerges in metaheuristics via memory and learning of algorithms. Intelligent metaheuristics that can learn and memorize maintain a single candidate solution or a population of solutions that provide the information acquired by the process, and the basis for making future decisions. The use of prior knowledge created by the adapted solutions can sometimes be interesting, innovative, and even competitive with human expertise (Koza, et al., 2003). Memory and learning abilities are among the main features that draw the line between human beings' excellence and other beings. The addition of intelliegence to Meta-RaPS is the main contribution of this dissertation.

CHAPTER 11

RESEARCH CONTRIBUTION

This research was inspired by the idea of creating intelligent algorithms that can learn the structure of optimization problems, memorize the search history and eventually produce high quality solutions. To reach this objective, different memory and learning approaches in the literature were presented to be integrated into Meta-RaPS, which is currently classified as a memoryless metaheuristic. Therefore, the contributions of this research are gathered around designing and implementing "intelligent" algorithms into Meta-RaPS by incorporating memory and learning techniques.

The first contribution is the introduction of Estimation of Distribution Algorithms (EDA) as a stochastic learning approach into Meta-RaPS. After investigating the EDA applications in the literature, a different but more efficient EDA form for this study's implementation was embedded into Meta-RaPS EDA. With EDA, the new algorithm could memorize the solution process by means of its memory matrix and use this information in making future decisions.

The second contribution is the utilization of a machine learning approach, named Q Learning. As in the case of EDA, Q Learning was also analyzed to reach its best performance for our application. This second version of Meta-RaPS with this new Q Learning form, called Meta-RaPS Q, could understand the structure of problems, and decide its next best step via a memory matrix to generate high quality solutions.

In the first two contributions, both of the proposed algorithms had separate memory matrices that should be trained to be able to extract the priorities needed for the

solution process by Meta-RaPS. However, in the third contribution, a new Meta-RaPS version that is intelligent in a very different way was implemented. In this proposed algorithm, Path Relinking (PR) was integrated into Meta-RaPS as a post-optimization method. The new Meta-RaPS PR algorithm, could learn the "good" attributes of the best solutions, and track them to reach better solutions without requiring any memory matrix.

The fourth contribution is made in the parameter tuning area that plays a key role in metaheuristics' performance. Tuning the parameters of the algorithm to their best values can be another challenging learning problem. Thefore, in the fourth proposed algorithm, Meta-RaPS with Adaptive Parameter (AP) has the ability to adaptively tune its two important parameters by creating a parameter memory matrix. While proceeding, Meta-RaPS AP could tune adaptively its parameters in each iteration, and with these best parameter settings the new algorithm could generate high quality solutions.

The fifth and last contribution is redesigning Meta-RaPS into a more "intelligent" metaheuristic as motivated by the successful memory and learning applications presented in this research. Although these proposed approches produced promising solutions, they still suffered from the high computational cost of the solution process. Together with the facts that most of solution time was consumed by the improvement phase, i.e. the local search algorithm within Meta-RaPS, and since PR did not require a memory matix and thus needed relatively lower amount of time, the new Meta-RaPS was redesigned by removing the improvement phase and integrating a more sophisticated PR approach. The new design of Meta-RaPS is renamed as Meta-RaPS V2.

In summary, the contributions presented in this research show that memory and learning mechanisms incorporated into a memoryless metaheuristic such as Meta-RaPS can result in a significant improvement to the metaheuristic's performance.

In these contributions, the memory and learning approaches that were integrated into Meta-RaPS were successfull at obtaining high quality solutions without affecting the main principles of Meta-RaPS, and therefore, they all can be conveniently applied in other population-based algorithms.

REFERENCES

Abraham, A. (2004) *Meta-Learning Evolutionary Artificial Neural Networks*, Neurocomputing Journal 56 c, pp. 1-38.

Agapie, A. (2010) *Estimation of Distribution Algorithms on Non-Separable Problems*, International Journal of Computer Mathematics, Vol. 87, No. 3, pp. 491-508.

Ahmadabadi, M.N. & Asadpour, M. (2002) *Expertness Based Cooperative Q-Learning*, IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics, Vol. 32, No. 1, pp. 66-76.

Ahn, C.W. & Ramakrishna, R.S. (2006) *Clustering-Based Probabilistic Model Fitting in Estimation of Distribution Algorithms*, IEICE Trans. Inf. & Syst. Vol. E89-D. No.1, pp. 381-383.

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, pp. 75-102.

Al-Salem, A. (2004) *Scheduling to Minimize Makespan on Unrelated Parallel Machines with Sequence Dependent Setup Times*, Engineering Journal of the University of Qatar, Vol, 17, pp. 177-187.

Alabas-Uslub, C. (2008) *A Self-Tuning Heuristic for A Multi-Objective Vehicle Routing Problem*, Journal of the Operational Research Society 59, pp. 988-996.

Alabas-Uslub, C. & Dengiz, B. (2011) *A Self-Adaptive Local Search Algorithm for the Classical Vehicle Routing Problem*, Expert Systems with Applications, Vol. 38, pp. 8990-8998.

Alpaydın, E. (2004), *Introduction to Machine Learning*, MIT Press, Cambridge, Massachusetts, London, England.

Alvarez-Valdes, R., Crespo, E., Tamarit, J.M. & Villa F. (2008) *GRASP and Path Relinking for Project Scheduling under Partially Renewable Resources*, European Journal of Operational Research, Vol. 189, pp. 1153-1170.

Anderson, J.R. (2000) *Learning and Memory: An Integrated Approach.* John Wiley & Sons, New York.

Andrade, D.V. & Resende, M.G.C. (2007) *GRASP with Evolutionary Path-Relinking*, AT&T Labs Research Technical Report TD-6XPTS7.

Andrecut, M. & Ali, M.K. (2001) *Q Learning in the Minority Game*, Physical Review E, Vol. 64, 067103, pp. 1-4.

Anghinolfi, D., Boccalatte, A., Paolucci, M. & Vecchiola, C. (2008) *Performance Evaluation of an Adaptive Ant Colony Optimization Applied to Single Machine Scheduling*, SEAL 2008, Springer-Verlag Berlin Heidelberg, pp. 411-420.

Arcus, A.L. (1966), *COMSOAL: A Computer Method Of Sequencing Operations For Assembly Lines*, The International Journal of Production Research, Vol.4, No. 4, pp. 259-277.

Arin, A., Rabadi, G. & Unal, R. (2011) *Comparative Studies on Design of Experiments for Tuning Parameters in a Genetic Algorithm for a Scheduling Problem*, International Journal of Experimental Design and Process Optimisation, Vol. 2, No.2, pp. 102-124.

Armentano, V.A., Shiguemoto, A.L. & Løkketangen, A. (2011) *Tabu search with Path Relinking for an Integrated Production-Distribution Problem*, Computers & Operations Research, v 38, n 8, pp.1199-209.

Arroyo, J.E.C., Santos, A.G., Silva, F.L.S., & Araújo A. F. (2008) *A GRASP with Path Relinking for the Single Machine Total Weighted Tardiness Problem*, 8th International Conference on Hybrid Intelligent Systems, pp. 726-731.

Balev, S., Yanev, N., Fréville, A. & Andonov, R. (2008) *A Dynamic Programming Based Reduction Procedure for the Multidimensional 0–1 Knapsack Problem*, European Journal of Operational Research 186, pp. 63-76.

Baluja, S. & Davies, S. (1997) *Using Optimal Dependency-Trees for Combinatorial Optimization: Learning the Structure of the Search Space*, Proceedings of the International Conference on Machine Learning, pp. 30-38.

Baluja, S., Pomerleau, D., & Jochem, T. (1994) *Towards Automated Artificial Evolution for Computer-Generated Images*, Connection Science, pp. 325-354.

Battarra, M., Golden, B.L. & Vigo, D. (2008) *Tuning a Parametric Clarke-Wright Heuristic via a Genetic Algorithm*, J Opl Res Soc 59, pp. 1568-1572.

Battarra, M., Öncan, T., Altınel, I.K., Golden, B., Vigo, D. & Phillips E. (2012) *An Evolutionary Approach for Tuning Parametric Esau and Williams Heuristics*, Journal of the Operational Research Society 63, pp. 368-378.

Battiti, R., Brunato, M. & Mascia, F. (2008) *Reactive Search and Intelligent Optimization*, Springer, New York, NY.

Battiti, R. & Tecchiolli, G. (1994) *The Reactive Tabu Search*, ORSA Journal on Computing, Vol. 6, No. 2, pp.126-140.

Battiti R. & Tecchiolli G. (1995) *Local Search with Memory: Benchmarking RTS*, OR-Spektrum 17, pp. 67-86.

Bean, J.C. (1994) *Genetic Algorithms and Random Keys for Sequencing and Optimization*, ORSA Journal on Computing 6, pp. 154-160.

Beasley, J.E. (1990) *OR-Library: Distributing Test Problems by Electronic Mail*, Journal of the Operational Journal Society, 41, pp. 170-181. http://people.brunel.ac.uk/~mastjjb/jeb/info.html.

Beausoleil, R.P., Baldoquin, G. & Montejo, R.A. (2008) *Multi-Start and Path Relinking Methods to Deal with Multiobjective Knapsack Problems*, Ann Oper Res 157, pp. 105-133.

Belarmino, A. & Laguna, M. (2006) *Fine-Tuning of Algorithms Using Fractional Experimental Designs and Local Search*, Operations Research, Vol. 54, No. 1, pp. 99-114.

Bellman, R. (1957) *Dynamic Programming*, Princeton University Press, Princeton, NJ.

Beni, G. (1988) *The Concept of Cellular Robotic System*, Proceedings IEEE Int. Symp. on Intelligent Control, Los Alamitos, CA, pp. 57-62.

Beni, G. & Wang, J. (1989) *Swarm Intelligence*, Proceedings Seventh Annual Meeting of the Robotics Society of Japan, pp. 425-428.

Bertsimas, D. & Demir, R. (2002) *An Approximate Dynamic Programming Approach to Multidimensional Knapsack Problem*, Management Science, 48(4), pp. 550-565.

Beyer, H.G. & Schwefel, H.P. (2002) *Evolution Strategies - A Comprehensive Introduction*, Nat Comput 1, pp. 3-52.

Bhatnagara, S. & Babu, M.K. (2008) *New Algorithms of the Q-Learning Type*, Automatica 44, pp. 1111-1119.

Binkley, K.J. & Hagiwara, M. (2007) *Applying Self-Adaptive Evolutionary Algorithms to Two-Dimensional Packing Problems Using A Four Corners' Heuristic*, European Journal of Operational Research 183, pp. 1230-1248.

Birattari, M. (2009) *Tuning Metaheuristics: A Machine Learning Perspective*, Studies in Computational Intelligence, Volume 197, Springer-Verlag Berlin Heidelberg.

Blum, C. & Li, X. (2008) *Swarm Intelligence in Optimization*, in: Blum, C. & Merkle, D. (eds.) Swarm Intelligence: Introduction and Applications, Springer-Verlag Berlin Heidelberg.

Bosman, P.A.N. & Grahl, J. (2008) *Matching Inductive Search Bias and Problem Structure in Continuous Estimation-Of-Distribution Algorithms*, European Journal of Operational Research 185, pp. 1246-1264.

Bonabeau, E., Dorigo, M. & Theraulaz, G. (1999) *Swarm Intelligence: From Natural to Artificial Systems*, Sante Fe Institute, Studies in the Sciences of Complexity, Oxford University Press, New York, Oxford.

Bonarini, A., Lazaric, A., Montrone, F. & Restelli, M. (2009) *Reinforcement Distribution in Fuzzy Q-Learning*, Fuzzy Sets and Systems 160, pp. 1420-1443.

Bookstein, A., Kulyukin, V.A. & Raita, T. (2002) *Generalized Hamming Distance*, Information Retrieval, 5, pp. 353-375.

Boussier, S., Vasquez, M., Vimont, Y., Hanafi, S. & Michelon, P. (2010) *A Multi-Level Search Strategy for The 0-1 Multidimensional Knapsack Problem*, Discrete Applied Mathematics 158, pp. 97-109.

Boyer, V., Elkihel, M. & El Baz, D. (2009) *Heuristics for the 0–1 Multidimensional Knapsack Problem*, European Journal of Operational Research 199, pp. 658-664.

Box, G.E.P. & Wilson, K.G. (1951) *On the Experimental Attainment of Optimum Conditions*, Journal of the Royal Statistical Society, B, Vol. 13, pp. 1-45.

Box, G.E.P., Hunter, W.G. & Hunter, J.S. (1978) *Statistics for Experimenters: An Introduction to Design, Data Analysis, and Model Building*, John Wiley & Sons.

Box, J.M. & Drapper, N.R. (1974) *On Minimum Point Second Order Designs*, Technometrics, 16, pp. 613-616.

Bozejko, W. (2010) *Parallel Path Relinking Method for the Single Machine Total Weighted Tardiness Problem with Sequence-Dependent Setups*, Journal of Intelligent Manufacturing, Vol. 21, No. 6, pp. 777-785.

Cail, X., Sowmya, A. & Trinder, J. (2006) *Learning Parameter Tuning for Object Extraction*, in: Narayanan, et al. (eds.) ACCV 2006, LNCS 3851, pp. 868-877, Springer-Verlag Berlin Heidelberg.

Cairon, P.V. & Dorigo, M. (1997) *Training and Delayed Reinforcements in Q-Learning Agents*, International Journal of Intelligent Systems, Vol. 12, pp. 695-724.

Castellani, U., Fusiello A., Gherardi, R. & Murino, V. (2007) *Automatic Selection of MRF Control Parameters by Reactive Tabu Search*, Image and Vision Computing 25, pp. 1824-1832.

Castro, D. D. & Mannor, S. (2010) *Adaptive Bases for Q-Learning*, 49th IEEE Conference on Decision and Control, Atlanta, GA, USA.

Chance, P. (2008) *Learning and Behavior: Active Learning*, Belmont, CA.

Chandran, B, Golden, B.L. & Wasil, E. (2003) *A Computational Study of Three Demon Algorithm Variants for Solving the Traveling Salesman Problem*, in: Bhargava, H.K. (ed.) Computational Modeling and Problem Solving in the Networked World, Kluwer Academic Publisher: NY, pp. 155-175.

Chen, A., Donovan, G., Sowmya, A. & Trinder J. (2002) *Inductive Clustering: Automatic Low Level Segmentation in High Resolution Images*, in: ISPRS Photogrammet, Comput. Vision Volume A., Graz, Austria.

Chen, C., Li, H.-X. & Dong, D. (2008) *Hybrid Control for Robot Navigation - A Hierarchical Q-Learning Algorithm*, IEEE Robotics & Automation Magazine, June, pp. 37-47.

Chen, S.-H., Chen, M.-C., Chang, P.-C., Zhang, Q. & Chen, Y.-M. (2010) *Guidelines for Developing Effective Estimation of Distribution Algorithms in Solving Single Machine Scheduling Problems*, Expert Systems with Applications, Vol. 37, pp. 6441-6451

Chen, T., Tang, K., Chen, G. & Yao, X. (2010) *Analysis of Computational Time of Simple Estimation of Distribution Algorithms*, IEEE Transactions on Evolutionary Computation, Vol. 14, No. 1, pp. 1-22.

Chiang, W. & Russell, R.A. (1997) *A Reactive Tabu Search Metaheuristic for the Vehicle Routing Problem with Time Windows*, University of Tulsa, INFORMS Journal on Computing, Vol.9, No.4, pp. 417-430.

Chiang, C.-W., Huang, Y.-Q., Lu, G.-Q. & Lin, S.-S. (2011) *Ant-inspired Search Techniques for Solving the Zero-one Knapsack Problem with Multiple Constraints*, International Journal of Advancements in Computing Technology, Volume 3, Number 4, pp. 242-255.

Choi, J., Realff, M.J. & Lee, J.H. (2007) *A Q-Learning-Based Method Applied to Stochastic Resource Constrained Project Scheduling with New Project Arrivals*, Int. J. Robust and Nonlinear Control, 17, pp. 1214-1231.

Chu, P.C. & Beasley, J.E. (1998) *A Genetic Algorithm for the Multidimensional Knapsack Problem*, Journal of Heuristics, Vol. 4, pp. 63-86.

Clausen, C. & Wechsler, H. (2000) *Quad-Q-Learning*, IEEE Transactions on Neural Networks, Vol. 11, No. 2, pp. 279-294.

Cleary, R. & O'Neill, M. (2005) *An Attribute Grammar Decoder for the 01 Multiconstrained Knapsack Problem*, LNCS 3448, pp. 34-45.

Cordeau, J.F., Gendreau, M., Laporte, G., Potvin, J.Y. & Semet, F. (2002) *A Guide to Vehicle Routing Heuristics*, Journal of the Operational Research Society 53, pp. 512-522.

Cotta, C. & Troya, J.M. (1998) *A Hybrid Genetic Algorithm for the 0-1 Multiple Knapsack Problem*, in: Artificial NN and GAs, Vol. 3. Springer, Heidelberg, pp 251-255.

Dammeyer, F. & Voss, S. (1993) *Dynamic Tabu List Management Using the Reverse Elimination Method*, Annals of Operations Research, Vol. 41, pp. 31-46.

Daniel, C. (1994) *Factorial One-Factor-at-a-Time Experiments*, The American Statistician, Vol. 48, No. 2, pp. 132-135.

De Bonet, J.S., Isbell, C.L. & Viola, P. (1997) *MIMIC: Finding Optima by Estimating Probability Densities*, Advances in Neural Information Processing Systems, Vol. 9, pp. 424-431.

De Jong, K. (2007) *Parameter Setting in EAs: A 30 Year Perspective*, in: Parameter Setting in Evolutionary Algorithms, Studies in Computational Intelligence, Springer, pp. 1-18.

Delorme, X., Gandibleux, X. & Rodriguez, J. (2004) *GRASP for Set Packing Problems*, European Journal of Operational Research, Vol. 153, pp. 564-580.

Deneubourg, J.-L., Goss, S., Franks, N.R. & Pasteels, J.M. (1989) *The Blind Leading the Blind: Modeling Chemically Mediated Army Ant Raid Patterns*, J. Insect Behav.,Vol. 2, pp. 719-725

Deng, Y. & Bard, J.F. (2011) *A Reactive GRASP with Path Relinking for Capacitated Clustering*, J Heuristics, Vol. 17, pp. 119-152.

DePuy, G.W., Moraga, R.J. & Whitehouse, G.E. (2005) *Meta-RaPS: A Simple and Effective Approach for Solving the Traveling Salesman Problem*, Transportation Research Part E: Logistics and Transportation Review, Vol. 41, No. 2, pp. 115-130.

DePuy, G.W. & Whitehouse, G.E. (2000) *Applying the COMSOAL Computer Heuristic to the Constrained Resource Allocation Problem*, Computers & Industrial Engineering, Vol. 38, pp. 413-422.

DePuy, G.W. & Whitehouse, G.E. (2001) *A Simple and Effective Heuristic for the Multiple Resource Allocation Problem*, International Journal of Production Research, Vol. 32, No. 4, pp. 24-31.

DePuy, G.W. & Whitehouse, G.E. (2001) *A Simple and Effective Heuristic for the Resource Constrained Project Scheduling Problem*, International Journal of Production Research, Vol. 39, No. 14, pp. 3275-3287.

DePuy, G.W., Whitehouse, G.E. & Moraga, R.J. (2001) *Meta-RaPS: A Simple and Efficient Approach for Solving Combinatorial Problems*, 29th International Conference on Computers and Industrial Engineering, November 1-3, Montreal, Canada, pp. 644-649.

Ding, N., Zhou, S.D. & Sun, Z.Q. (2008) *Histogram-Based Estimation of Distribution Algorithm: A Competent Method for Continuous Optimization*, Journal of Computer Science and Technology, Vol. 23, No. 1, pp. 35-43.

Dorigo, M. (1992) *Optimization, Learning and Natural Algorithms*, Ph.D. Dissertation, Politecnico di Milano, Italy.

Dorigo, M. & Stützle T. (2004) *Ant Colony Optimization*, MIT Press, Massachusetts.

Dréo, J., Aumasson, J.-P., Tfaili, W. & Siarry, P. (2007) *Adaptive Learning Search, A New Tool to Help Comprehending Metaheuristics*, International Journal on Artificial Intelligence Tools, Vol. 16, No. 3.

Dréo, J., Pétrowski, A., Siarry, P. & Taillard, E. (2006) *Metaheuristics for Hard Optimization*, Springer, Berlin, Heidelberg.

Drexl, A. (1988) *A Simulating Annealing Approach to Multiconstraint Zero-One Knapsack Problem*, Computing, Vol. 40, pp. 1-8.

Duarte, A., Martí, R. & Gortazar, F. (2011) *Path Relinking for Large-Scale Global Optimization*, Soft Computing, Vol. 15, No. 11, pp. 2257-2273.

Eiben, A.E., Hinterding, R., & Michalewicz, Z. (1999) *Parameter Control in Evolutionary Algorithms*, IEEE Transactions on Evolutionary Computation, Vol. 3, No. 2, pp. 124-141.

Eiben, A.E. & Smith, J.E. (2003) *Introduction to Evolutionary Computing*, Springer.

El-Fallahi, A., Martí, R. & Lasdon, L. (2006) *Path Relinking and GRG for Artificial Neural Networks*, European Journal of Operational Research, Vol. 169, pp. 508-519.

Engelbrecht, A.P. (2007) *Computational Intelligence: An Introduction*, John Wiley & Sons, second edition.

Er, M.J. & Deng, C. (2004) *Online Tuning of Fuzzy Inference Systems Using Dynamic Fuzzy Q-Learning*, IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics, Vol. 34, No. 3, pp. 1478-1489.

Favuzza, S., Graditi, G. & Sanseverino, E.R. (2006) *Adaptive and Dynamic Ant Colony Search Algorithm for Optimal Distribution Systems Reinforcement Strategy*, Applied Intelligence, Vol. 24, pp. 31-42.

Festa, P. & Resende, M.G.C. (2011) GRASP: *Basic Components and Enhancements*, Telecommun Syst, Vol. 46, pp. 253-271.

Feo, T.A. & Resende, M.G.C. (1989) *A Probabilistic Heuristic for a Computationally Difficult Set Covering Problem*, Operations Research Letters, Vol. 8, pp. 67-71.

Fığlalı, N., Özkale, C., Engin, O. & Fığlalı, A. (2009) *Investigation of Ant System Parameter Interactions by Using Design of Experiments for Job-Shop Scheduling Problems*, Computers & Industrial Engineering, Vol. 56, pp. 538-559.

Fleszar, K. & Hindi, K.S. (2009) *Fast, Effective Heuristics for the 0-1 Multi-Dimensional Knapsack Problem*, Computers & Operations Research, Vol. 36, pp. 1602-1607.

Fogel, D.B. (1995) *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence,* IEEE Press, Piscataway, NJ.

Fogel, D.B., Fogel, L.J. & Atma, J.W. (1991) *Meta-Evolutionary Programming*, in: Proceedings of 25th Conference on Signals, Systems & Computers, pp. 540-545.

Fogel, L.J., Owens, A. J. & Walsh, M.J. (1966) *Artificial Intelligence through Simulated Evolution*, Wiley.

Fogel, L.J. (1962) *Toward Inductive Inference Automata*, in Proceedings of the International Federation for Information Processing Congress, Munich, pp. 395-399.

Fréville, A. (2004) *The Multidimensional 0–1 Knapsack Problem: An Overview*, European Journal of Operational Research, Vol. 155, pp. 1-21.

Fréville, A. & Hanafi S. (1998) *An Efficient Tabu Search Approach for the 0-1 Multidimensional Knapsack Problem*, European Journal of Operational Research, Vol. 106, pp. 659-675.

Fréville, A. & Hanafi, S. (2005) *The Multidimensional 0-1 Knapsack Problem - Bounds and Computational Aspects*, Ann. Oper. Res., Vol. 139, pp. 195-227.

Fréville, A. & Plateau, G. (1990) *Hard 0-1 Multiknapsack Test Problems for Size Reduction Methods*, Investigation Operativa, Vol. 1, pp. 251-270.

Fréville, A. & Plateau, G. (1994) *An Efficient Preprocessing Procedure for the Multidimensional 0-1 Knapsack Problem*, Discrete Applied Mathematics, Vol. 49, pp. 189-212.

Fuchida, T., Aung, K.T. & Sakuragi, A. (2010) *A Study of Q-Learning Considering Negative Rewards*, Artif. Life Robotics, Vol. 15, pp. 351-354.

Gabrel, V., Knippel, A. & Minoux, M. (1999) *Exact Solutions of Multicommodity Network Optimization Problems with General Step Cost Functions*, Oper. Res. Lett., Vol. 25, pp. 15-23.

Gallardo, J.E., Cotta, C. & Fernández, A.J. (2007) *On the Hybridization of Memetic Algorithms With Branch-and-Bound Techniques*, IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics, Vol. 37, No. 1, pp. 77-83.

Gallardom J.E., Cottam C., & Fernandez, A.J. (2009) *Exact, Metaheuristic, and Hybrid Approaches to Multidimensional Knapsack Problems: Optimization Techniques for Solving Complex Problems*, John Wiley & Sons, Hoboken, New Jersey.

Gendreau, M. & Potvin, J.-Y. (2005) *Tabu Search*, in: Burke, E.K. & Kendall, G. (eds.) Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques, Springer, New York, NY.

Gendreau, M. & Potvin, J.-Y. (2007) *Metaheuristics: A Canadian Perspective*, CIRRELT-2007-60, Canada.

Gao, Y. & Culberson, J. (2005) *Space Complexity of Estimation of Distribution Algorithms*, Evolutionary Computation, Vol. 13, No. 1, pp. 125-143.

Garcia, C. & Rabadi, G. (2011) *A Meta-RaPS Algorithm for Spatial Scheduling with Release Times*, Int. J. Planning and Scheduling, Vol. 1, Nos. 1/2, pp. 19-31.

Garey, M.R. & Johnson, D. J. (1979) *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco.

Gavish, B. & Pirkul, H. (1982) *Allocation of Databases and Processors*, in: Akola, J. (ed.) A Distributed Data Processing, Management of Distributed Data Processing, North-Holland, Amsterdam, pp. 215-231.

Gavish, B. & Pirkul, H. (1985) *Efficient Algorithms for Solving Multi-Constraint Zero-One Knapsack Problems to Optimality*, Mathematical Programming, Vol. 31, pp. 78-105.

Gilmore, P. & Gomory, R. (1966) *The Theory and Computation Of Knapsack Functions*, Operations Research, Vol. 14, pp. 1045-1074.

Glover, F. (1977) *Heuristics for Integer Programming Using Surrogate Constraints*, Decision Sciences, Vol. 8, pp.156-166.

Glover, F. (1989) *Tabu Search: Part I*, ORSA Journal on Computing, Vol. 1, No. 3, pp. 190-206.

Glover, F. (1996) *Tabu Search and Adaptive Memory Programing - Advances, Applications and Challenges*, in: Barr, R.S., Helgason, R.V. & Kennington, J.L. (eds.) Interfaces in Computer Science and Operations Research, Kluwer.

Glover, F. & Kochenberger, G. (1996) *Critical Event Tabu Search for Multidimentional Knapsack Problems*, in: Osman, I.H. & Kelly, J.P. (eds.) Meta-heuristics: Theory and applications, Dordecht Kluwer Academics Publishers, pp. 407-427.

Glover, F. & Laguna, M. (1993) *Tabu Search*, in: Reeves, C.R. (ed.) Modern Heuristic Techniques for Combinatorial Problems, Blackwell Publishing, Oxford, pp. 70-150.

Glover ,F. & Laguna, M. (1997) *Tabu Search*, Kluwer Academic Publishers, Boston.

Glover, F., Laguna, M. & Marti, R. (2000) *Fundamentals of Scatter Search and Path Relinking*, Control and Cybernetics, Vol. 29, No. 3, pp. 653-684.

Glover, F., Laguna, M. & Marti, R. (2003) *Scatter Search and Path Linking*, in: Glover F. & Kochenberger, G.A. (eds.) Handbook of Metaheuristics, Kluwer Academic Publishers.

Goldberg, D.E. (2002) *The Design of Innovation: Lessons from and for Competent Genetic Algorithms*, Kluwer.

Golden, B.L., Pepper, J. & Vossen, T. (1998) *Using Genetic Algorithms for Setting Parameter Values in Heuristic Search*, in: Dagli, C., Akay, A., Buczak, M., Ersoy, O. & Fernandez, B. (eds.) Intelligent Engineering System through Artificial Neural Networks. Vol. 8, ASME Press: NY, pp 239-245.

Gomes, F.C., Pardalos, P., Oliveira, C.S. & Resende, M.G.C.(2001) *Reactive GRASP with Path Relinking for Channel Assignment in Mobile Phone Networks*, Proceedings of the 5th International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications, pp. 60-67.

Gosavi, A (2009) *Reinforcement Learning: A Tutorial Survey and Recent Advances*, INFORMS Journal on Computing, Vol. 21, No. 2, pp. 178-192.

Gosavi, A (2009) *On Step Sizes, Stochastic Shortest Paths, and Survival Probabilities in Reinforcement Learning*, Conference Proceedings of the Winter Simulation Conference.

Gong, Q., Zhou, Y. & Luo, Q. (2011) *Hybrid Artificial Glowworm Swarm Optimization Algorithm for Solving Multi-dimensional Knapsack Problem*, Procedia Engineering Vol. 15, pp. 2880-2884.

Grahl, J. (2007) *Estimation of Distribution Algorithms in Logistics: Analysis, Design, and Application*, Ph.D. Dissertation, Mannheim University, Dortmund.

Grasse, P.-P. (1984) *Termitologia*, Tome II, Fondation des Societes, Construction, Paris, Masson.

Grasse, P.-P. (1959) *La Reconstruction du Nid et les Coordinations Inter-Individuelles chez Bellicositerm, et Natalensis et Cubitermes sp*, La theorie de la Stigmergie: Essai d'interpretation du Comportement des Termites Constructeurs, Insect. Soc., Vol, 6, pp. 41-80.

Guo, M., Liu, Y. & Malec, J. (2004) *A New Q-Learning Algorithm Based on the Metropolis Criterion*, IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics, Vol. 34, No. 5, pp. 214-2143.

Hackwood, S. & Beni, G. (1992) *Self-Organization of Sensors for Swarm Intelligence*, in Proceedings IEEE 1992 International Conference on Robotics and Automation, IEEE Computer Society Press, Los Alamitos, CA, pp. 819-829.

Hagen, T.S. & Kröse, B. (2003) *Neural Q-Learning*, Neural Comput & Applic, Vol. 12, pp. 81-88.

Haken, H. (1983) *Synergetics*, Berlin, Springer-Verlag.

Handa, H. (2007) *The Effectiveness of Mutation Operation in the Case of Estimation of Distribution Algorithms*, BioSystems, Vol. 87, pp. 243-251.

Hanafi, S. & Fréville, A. (1998) *An Efficient Tabu Search Approach for the 0–1 Multidimensional Knapsack Problem*, Eur. J. Oper. Res., Vol. 106, pp. 659-675.

Hanafi, S., Fréville, A., & El Abdellaqui, A. (1996) *Comparison of Heuristics for the 0-1 Multidimensional Knapsack Problem*, in: Osman, I.H. & Kelly, J.P. (eds.) Meta-heuristics: Theory and Applications, Dordecht: Kluwer Academics Publishers, pp. 449-465.

Harik, G.R., Lobo, F.G. & Goldberg, D.E. (1997) *The Compact Genetic Algorithm*, IlliGAL Report No. 97006, University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, Urbana, IL.

Haul, C. & Voss, S. (1997) *Using Surrogate Constraints in Genetic Algorithms for Solving Multidimensional Knapsack Problems*, in: Woodruff, D.L. (ed.) Advances in Computational and Stochastic Optimization, Logic Programming, and Heuristic Search, Interfaces in Computer Science and Operation Research, Dordecht Kluwer Academic Publishers, pp. 235-251.

Hauschild, M. & Pelikan, M. (2011) *An Introduction and Survey of Estimation of Distribution Algorithms*, Swarm and Evolutionary Computation, Vol. 1, pp. 111-128.

Hembecker, F., Lopes, H.S. & Godoy, Jr.W. (2007) *Particle Swarm Optimization for the Multidimensional Knapsack Problem,* LNCS Adapt Nat Comput Algorithms, Vol. 4431, pp. 358-365.

Hepdogan, S., Moraga, R.J., DePuy, G.W. & Whitehouse, G.E. (2008) *Nonparametric Comparison of Two Dynamic Parameter Setting Methods in a Meta-Heuristic Approach*, Systemics, Cybernetics and Informatics, Vol. 5, No. 5, pp. 46-52.

Hepdogan, S., Moraga, R.J., DePuy, G.W. & Whitehouse, G.E. (2009) *A Meta-RaPS For The Early/Tardy Single Machine Scheduling Problem*, International Journal of Production Research, Vol. 47, No. 7, pp. 1717-1732.

He, Y., Qiu, Y. & Liu, G. (2006) *A Tabu Search Approach with Double Tabu-List for Multidimensional Knapsack Problems*, IJCSNS International Journal of Computer Science and Network Security, Vol.6, No.5A, pp. 87-92.

Hill, R.R., Cho, Y.K. & Moore, J.T. (2012) *Problem Reduction Heuristic for the 0–1 Multidimensional Knapsack Problem*, Computers & Operations Research, Vol. 39, pp. 19-26.

Hirashima, Y., Iiguni, Y., Inoue, A. & Masuda, S. (1999) *Q-Learning Algorithm Using an Adaptive-Sized Q-Table*, Proceedings of the 38th Conference on Decision & Control, Phoenix, Arizona USA.

Holland, J.H. (1962) *Outline for a Logical Theory of Adaptive Systems*, Journal of the ACM, Vol. 3, pp. 297-314.

Holland, J.H. (1975) *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, MI.

Hongcheng, L., Liang, G. & Quanke, P. (2011) *A Hybrid Particle Swarm Optimization with Estimation of Distribution Algorithm for Solving Permutation Flowshop Scheduling Problem*, Expert Systems with Applications, Vol. 38, No. 4, pp. 4348-60.

Huang, X., Jia, P. & Liu, B. (2010) *Controlling Chaos by an Improved Estimation of Distribution Algorithm*, Mathematical and Computational Applications, Vol. 15, No. 5 SPEC.ISSUE, pp. 866-871.

Hwang, K.-S., Lin, H.-Y., Hsu, Y.-P. & Yu, H.-H. (2011) *Self-Organizing State Aggregation for Architecture Design of Q-Learning*, Information Sciences, Vol. 181, pp. 2813-2822

Ibaraki, T., Nonobe, K. & Yagiura, M. (2005) *Metaheuristics: Progress as Real Problem Solvers*, Springer Science+Business Media, Inc., USA.

Ide, A. & Yasuda, K. (2005) *A Basic Study of Adaptive Particle Swarm Optimization*, Electrical Engineering in Japan, Vol. 151, No. 3, pp. 41-49.

Jaakkola, T., Jordan, M.I. & Singh, S.P. (1994) *On the convergence of Stochastic Iterative Dynamic Programming Algorithms*, Neural Computation, Vol. 6, pp. 1185-1201.

Jaeggi, D.M., Parks, G.T., Kipouros, T. & Clarkson, P.J. (2008) *The Development of a Multi-Objective Tabu Search Algorithm for Continuous Optimisation Problems*, European Journal of Operational Research, Vol. 185, No. 3, pp.1192-212.

James, R.J.W. & Nakagama, Y. (2005) *Enumeration Methods for Repeatedly Solving Multidimensional Knapsack Sub-problems*, IEICE Trans. Inf.&Syst, Vol. E88-D, pp. 2329-2340.

Jarboui, B., Eddaly, M. & Siarry, P. (2009) *An Estimation of Distribution Algorithm for Minimizing the Total Flowtime in Permutation Flowshop Scheduling Problems*, Computers & Operations Research, Vol. 36, No. 9, pp. 2638-2646.

Jaszkiewicz, A. & Zielniewicz, P. (2009) *Pareto Memetic Algorithm with Path Relinking for Bi-Objective Traveling Salesperson Problem*, European Journal of Operational Research, Vol. 193, No. 3, pp. 885-890.

Jeon, S.M., Kim, K.H. & Kopfer, H. (2011) *Routing Automated Guided Vehicles in Container Terminals through the Q-Learning Technique*, Logist. Res., Vol. 3, pp. 19-27.

Jiang, S., Ziver, A.K., Carter, J.N., Pain, C.C., Goddard, A. J. H., Franklin, S. & Phillips, H.J. (2006) *Estimation of Distribution Algorithms for Nuclear Reactor Fuel Management Optimisation*, Annals of Nuclear Energy, Vol. 33, pp. 1039-1057.

Juang, C.-F. & Lu, C.-M. (2009) *Ant Colony Optimization Incorporated with Fuzzy Q-Learning for Reinforcement Fuzzy Control*, IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans, Vol. 39, No. 3, pp. 597-608.

J'unior, F.C.D.L., Melo, J.D.D. & Neto, A.D.D. (2008) *Using the Q-learning Algorithm in the Constructive Phase of the GRASP and Reactive GRASP Metaheuristics*, International Joint Conference on Neural Networks (IJCNN 2008), pp. 4169-4176.

Kaelbling, L.P., Littman, M.L. & Moore, A.W. (1996) *Reinforcement Learning: A Survey*, Journal of Artificial Intelligence Research, Vol. 4, pp. 237-285.

Kamali, K., Jiang, L.J. & Yen, J. (2007) *Q-Learning and Genetic Algorithms to Improve the Efficiency of Weight Adjustments for Optimal Control and Design Problems*, Transactions of the ASME, Vol. 7, December, pp. 302-308.

Kaplan, S., Arin, A. & Rabadi, G. (2010) *Meta-RaPS Algorithm for the Aerial Refueling Scheduling Problem*, International Conference of Modeling and Simulation (MODSIM), Virginia, USA.

Kaplan, S. & Rabadi, G. (in press) *A Simulated Annealing and Meta-RaPS Algorithms for the Aerial Refueling Scheduling Problem with Due Date-to-Deadline Windows and Release Time*, Engineering Optimization.

Kato, K. & Sakawa, M. (2003) *Genetic Algorithms with Decomposition Procedures for Multidimensional 0-1 Knapsack Problems with Block Angular Structures*, IEEE Transactions on Systems, Man, and Cybernetics—Part B: Cybernetics, Vol. 33, No. 3, pp. 410-419.

Kazdin, A.E. (2000) *Encyclopedia of Psychology*, Oxford University Press, USA.

Kellerer, H., Pferschy, U. & Pisinger, D. (2004) Knapsack Problems, Springer.

Kennedy, J., Eberhart, R.C. & Shi, Y. (2001) *Swarm Intelligence: Collective, Adaptive*, Morgan Kaufmann, San Francisco, CA.

Kesner, R.P. (1998) *Neurobiology of Learning and Memory*, in: Martinez, Jr.J.L. & Kesner, R.P. (eds.) Neurobiological Views of Memory, Academic Press, California.

Khuri, S., Back, T. & Heitkotter, J. (1994) *The Zero/One Multiple Knapsack Problem and Genetic Algorithms*, Proceedings of the ACMsymposium of Applied Comp, pp. 188-193.

Kivijarvi, J., Franti, P. & Nevalainen, O. (2003) *Self-Adaptive Genetic Algorithm for Clustering*, Journal of Heuristics, Vol. 9, pp. 113-129.

Klopf, A.H. (1972) *Brain Function and Adaptive Systems - A Heterostatic Theory*, Technical Report AFCRL - 72 - O164, Air Force Cambridge Research Laboratories, Bedford, MA.

Kong, M. (2007) *A New Ant Colony Optimization Algorithm for the Multidimensional Knapsack Problem*, Computers and Operations Research, doi:10.1016/j.cor.2006.12.029.

Koza, J.R. (1992) *Genetic Programming*, MIT Press, Cambridge, MA.

Koza, J.R., Keane, M.A., Streeter, M.J., Mydlowec, W., Yu, J. & Lanza, G. (2003) *Genetic Programming IV: Routine Human-Competitive Machine Intelligence*, Springer.

Kramer, O. (2008) *Self-Adaptive Heuristics for Evolutionary Computation*, Studies in Computational Intelligence, Vol. 147, Springer-Verlag Berlin Heidelberg.

Kramer, O. (2010) *Evolutionary Self-Adaptation: A Survey of Operators and Strategy Parameters*, Evol. Intel., Vol. 3, pp. 51-65.

Kramer, O., Gloger, B. & Goebels, A. (2007) *An Experimental Analysis of Evolution Strategies and Particle Swarm Optimisers using Design of Experiments*, GECCO'07, pp. 674-481.

Laguna, M. & Martí, R. (1999) *GRASP And Path Relinking for the 2-Layer Straight Line Crossing Minimization*, INFORMS Journal on Computing, Vol. 11, pp. 44-52.

Lan, G. & DePuy, G.W. (2006) *On the Effectiveness of Incorporating Randomness and Memory into A Multi-Start Metaheuristic with Application to the Set Covering Problem*, Computers & Industrial Engineering, Vol. 51, pp. 362-374.

Lan, G., DePuy, G.W. & Whitehouse, G.E. (2007) *An Effective and Simple Heuristic for the Set Covering Problem*, European Journal of Operational Research, Vol. 176, pp. 1387-1403

Langdon, W. B. & Poli, R. (2002) *Foundations of Genetic Programming*, Springer, Berlin.

Langlois, M. & Sloan, R.H. (2010) *Reinforcement Learning via Approximation of the Q-Function*, Journal of Experimental & Theoretical Artificial Intelligence, Vol. 22, No. 3, pp. 219-235.

Laporte, G., Gendreau M., Potvin J.Y. & Semet F. (2000) *Classical and Modern Heuristics for the Vehicle Routing Problem*, Int Trans Opl Res, Vol. 7, pp. 285-300.

Lee, J.W., Park, J., O, J., Lee, J. & Hong, E. (2007) *A Multiagent Approach to Q-Learning for Daily Stock Trading*, IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans, Vol. 37, No. 6, pp. 864-877.

Li, G., Li, M., Azarm, S., Al Hashimi, S., Al Ameri, T. & Al Qasas, N. (2009) *Improving Multi-Objective Genetic Algorithms with Adaptive Design of Experiments and Online Metamodeling*, Struct Multidisc Optim, Vol. 37, pp. 447-461.

Lin, L., Xie, H., Zhang, D. & Shen, L. (2010) *Supervised Neural Q_learning based Motion Control for Bionic Underwater Robots*, Journal of Bionic Engineering, Vol. 7 Suppl., pp.177-184.

Lima, F.C., Lobo, G.F., Pelikan, M. & Goldberg, D.E. (2011) *Model Accuracy in the Bayesian Optimization Algorithm*, Soft Comput, Vol. 15, pp. 1351-1371.

Lima, F.C., Melo, J.D. & Neto, A.D.D. (2007) *Proposal for Improvement of GRASP Metaheuristic and Genetic Algorithm Using the Q-Learning Algorithm*, Seventh International Conference on Intelligent Systems Design and Applications, IEEE Computer Society, pp. 465-470.

Liu, Y.-F. & Liu, S.-Y. (2011) *A Hybrid Discrete Artificial Bee Colony Algorithm for Permutation Flowshop Scheduling Problem*, Appl. Soft Comput. J., doi:10.1016/j.asoc.2011.10.024

Lokketangen, A., & Glover, F. (1998) *Solving Zero-One Mixed Integer Programming Problems Using Tabu Search*, European Journal of Operations Research, Vol. 106, pp. 624-658.

Lorie, J.H. & Savage, L.J. (1955) *Three Problems in Capital Rationing*, J. Bus., Vol. 28, pp. 229-239.

Lozano, J.A., Larrañaga, P., Inz, I. & Bengoetxea, E. (2006) *Evolutionary Computation: Towards a New Advances in the Estimation of Distribution Algorithms*, Springer-Verlag Berlin Heidelberg.

Magazine, M.J. & Oguz, O. (1984) *A Heuristic Algorithm for the Multidimensional Zero-One Knapsack Problem*, European Journal of Operational Research, Vol. 16, pp. 319-326.

Manne, A.S. & Markowitz, H.M. (1957) *On the Solution of Discrete Programming Problems*, Econometrica, Vol. 25, pp. 84-110.

Martello, S. & Toth, P. (1990) *Knapsack Problems: Algorithms and Computer Implementations*, John Wiley & Sons, New York

Martello, S., Pisinger, D. & Toth, P. (1999) *Dynamic Programming and Strong Bounds for the 0-1 Knapsack Problem*, Management Science, Vol. 45, pp. 414-424.

Martello, S., Pisinger, D. & Toth, P. (2000) *New Trends in Exact Algorithms for the 0-1 Knapsack Problem*, European Journal of Operational Research, Vol.123, No.2, pp. 325-332.

Martello, S. & Toth, P. (1988) *A New Algorithm for the 0-1 Knapsack Problem*, Management Science, Vol. 34, pp. 633-644.

Mateus, G.R., Resende, M.G.C. & Silv,a R.M.A. (2011) *GRASP with Path-Relinking for the Generalized Quadratic Assignment Problem*, Journal of Heuristics, Vol. 17, No. 5, pp. 527-565.

Maoguo, G., Licheng, J., Wenping, M. & Shuiping, G. (2007) *Solving Multidimensional Knapsack Problems By An Immune-Inspired Algorithm*, Evolutionary Computation, CEC 2007, pp. 3385-3391.

Martello, S. & Toth, P. (1990) *Knapsack problems: Algorithms and Computer Implementations*, John Wiley & Sons, Chichester, England.

Martí, L., Garca, J., Berlanga, A., Coello, C.A. & Molina, J.M. (2011) *MB-GNG: Addressing Drawbacks in Multi-Objective Optimization Estimation of Distribution Algorithms*, Operations Research Letters, Vol. 39, No. 2, pp. 150-154.

Martí, R., Montes, F. & El-Fallahi, A. (2005) *Approximating Unknown Mappings: An Experimental Evaluation*, Journal of Heuristics, Vol. 11, pp. 219-232.

Meier, H., Christofides, N. & Salkin, G. (2001) *Capital Budgeting Under Uncertainty - An Integrated Approach Using Contingent Claims Analysis and Integer Programming*, Oper. Res., Vol. 49, pp. 196-206.

Mercer, R.E. & Sampson, J.R. (1978) *Adaptive Search Using A Reproductive Metaplan*, Kybernetes, Vol. 7, pp. 215-228.

Merkle, D. & Middendorf, M. (2005) *Swarm Intelligence*, in: Burke, E.K. & Kendall, G. (eds.) Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques, Springer, New York, NY.

Millán, J.D.R., Posenato D. & Dedieu E. (2002) *Continuous-Action Q-Learning*, Machine Learning, Vol. 49, pp. 247-265,

Miquélez, T., Bengoetxea, E., Mendiburu, A. & Larrañaga, P. (2007) *Combining Bayesian Classfiers and Estimation of Distribution Algorithms for Optimization in Continuous Domains*, Connection Science, Vol. 19, No. 4, pp. 297-319.

Mitchell, T.M. (1997) *Machine Learning*, McGraw-Hill, New York

Monekosso, N. & Remagnino, P. (2004) *The Analysis and Performance Evaluation of the Pheromone-Q-Learning Algorithm*, Expert Systems, Vol. 21, No. 2, pp. 80-91.

Montgomery, D.C. & Runger, G.C. (2003), *Applied Statistics Applied Statistics for Engineers*, Third Edition, John Wiley & Sons, Inc., New York, NY.

Moraga, R.J. (2002) *Meta-RaPS: An Effective Solution Approach for Combinatorial Problems*, Ph.D. Dissertation, University of Central Florida, Orlando, FL.

Moraga, R.J. (2009) *Meta-RaPS: Optimization Methods Class Notes*, Northern Illinois University, IL.

Moraga, R.J., DePuy G.W. & Whitehouse G.E. (2005) *Meta-RaPS Approach For The 0-1 Multidimensional Knapsack Problem*, Computers & Industrial Engineering, No. 48, pp. 83-96.

Moraga, R.J., DePuy, G.W. & Whitehouse, G.E. (2006) *Metaheuristics: A Solution Methodology for Optimization Problems, Handbook of Industrial and Systems Engineering*, CRC Press, New York.

Mumford, C.L. & Jain, L.C. (2009) *Computational Intelligence: Collaboration, Fusion and Emergence*, Springer-Verlag Berlin Heidelberg,

Mühlenbein, H. (2008) *Convergence of Estimation of Distribution Algorithms for Finite Samples*, Technical Report, Fraunhofer Institut Autonomous intelligent Systems, Sankt Augustin.

Mühlenbein, H. & Mahnig, T. (1999) *FDA - A Scalable Evolutionary Algorithm for the Optimization of Additively Decomposed Functions*, Evolutionary Computation, Vol. 7, pp. 353-376.

Mühlenbein, H., Mahnig, T. & Ochoa, A. (1999) *Schemata, Distributions and Graphical Models in Evolutionary Optimization*, Journal of Heuristics, Vol, 5, No. 2, pp. 213-247.

Mühlenbein, H. & Paaß, G.H. (1996) *From Recombination of Genes to the Estimation of Distributions I. Binary Parameters*, in: Eiben, A., Bäck, T., Shoenauer, M. & Schwefel, H. (eds.) Parallel Problem Solving from Nature, Springer Verlag, Berlin, pp. 178-187.

Nascimento, M.C.V., Resende, M.G.C. & Toledo, F.M.B. (2010) *GRASP Heuristic with Path-Relinking for the Multi-Plant Capacitated Lot Sizing Problem*, European Journal of Operational Research, Vol. 200, No. 3, pp.747-54.

Nasiri, M.M. & Kianfar, F. (2011) *A Hybrid Scatter Search for the Partial Job Shop Scheduling Problem*, Int J Adv Manuf Technol., Vol. 52, pp. 1031-1038.

Nasiri, M.M. & Kianfar, F. (2012) *A Guided Tabu Search/Path Relinking Algorithm for the Job Shop Problem*, Int J Adv Manuf Technol., Vol. 58, pp. 1105-1113.

Nicolis, G. & Prigogine, I. (1977) *Self-Organization in Non-Equilibrium Systems*, Wiley & Sons, New York, NY.

Nie, J. & Haykin, S. (1999) *A Q-Learning-Based Dynamic Channel Assignment Technique for Mobile Communication Systems*, IEEE Transactions on Vehicular Technology, Vol. 48, No. 5, pp. 1676-1687.

Ollington, R.B. & Vamplew, P.W. (2005) *Concurrent Q-Learning: Reinforcement Learning for Dynamic Goals and Environments*, International Journal of Intelligent Systems, Vol. 20, pp. 1037-1052.

Ormrod, J.E. (2008) *Human Learning*, Pearson Education, Inc., New York.

Osorio, M.A., Glover, F. & Hammer, P. (2003) *Cutting and Surrogate Constraint Analysis for Improved Multidimensional Knapsack Solutions*, Annals of Operations Research, Vol. 117, pp. 71-93.

Osorio, M.A. & Cuaya, G. (2005) *Hard Problem Generation for MKP*, Revista Investigacion Operacional, Vol. 26, No 3, pp. 212-218.

Pacheco, J. & Martí, R. (2006) *Tabu Search for A Multi-Objective Routing Problem*, Journal of the Operational Research Society, Vol. 57, pp. 29-37.

Panigrahi, B.K., Shi, Y. & Lim, M.-H. (2011) *Handbook of Swarm Intelligence: Concepts, Principles and Applications*, Springer-Verlag Berlin Heidelberg.

Park, K.-H., Kim, Y.-J. & Kim, J.-H. (2001) *Modular Q-Learning Based Multi-Agent Cooperation for Robot Soccer*, Robotics and Autonomous Systems, Vol. 35, pp. 109-122.

Patricio, M.A., García, J., Berlanga, A. & Molina, J. M. (2009) *Visual Data Association for Real-Time Video Tracking Using Genetic and Estimation of Distribution Algorithms*, International Journal of Imaging Systems and Technology, Vol. 19, No. 3, pp. 199-207.

Pedrycz, W. (1997) *Computational Intelligence: An Introduction*, CRC Press.

Pelikan, M., Goldberg, D.E. & Cantú-Paz E. (2000) *Linkage Problem, Distribution Estimation, and Bayesian Networks*, Evolutionary Computation, Vol. 8, pp. 311-341.

Pelikan, M., Goldberg, D.E. & Lobo, F. (2002) *A Survey of Optimization by Building and Using Probabilistic Models*, Computational Optimization and Applications, Vol. 21, pp. 5-20.

Pelikan, M. & Mühlenbein, H. (1999) *The Bivariate Marginal Distribution Algorithm*, in: Roy, R., Furuhashi, T. & Chawdhry, P. K., (eds.) *Advances in Soft Computing - Engineering Design and Manufacturing*, pp. 521-535, Springer-Verlag, London.

Pelikan, M., Tsutsui, S. & Kalapala, R. (2007) *Dependency Trees, Permutations, and Quadratic Assignment Problem*, Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation, GECCO'07, ACM, New York, NY, USA, pp. 629-629.

Pepper, J., Golden, B. & Wasil, E. (2002) *Solving the Traveling Salesman Problem with Annealing-Based Heuristics: A Computational Study*, IEEE Trans Syst Man Cybern., Vol. A 32, pp. 72-77.

Pessoa, L.S., Resende, M.G.C. & Ribeiro, C.C. (2012) *A Hybrid Lagrangean Heuristic With Grasp and Path-Relinking for Set K-Covering*, Computers and Operations Research, doi:10.1016/j.cor.2011.11.018.

Petersen, C.C. (1967) *Computational Experience with Variants of the Balas Algorithm Applied to the Selection of R&D Projects*, Management Science, Vol. 13, No. 9, pp. 736-750.

Pirkul, H. (1987) *A Heuristic Solution Procedure for the Multiconstraint Zero-One Knapsack Problem*, Naval Research Logistics, Vol. 34, pp. 161-172.

Pisinger, D. (1995) *Algorithms for Knapsack Problems*, Ph.D. Dissertation, DIKU, University of Copenhagen, Report 95/1.

Pisinger, D. (1995) *An Expanding-Core Algorithm for the Exact 0-1 Knapsack Problem*, European Journal of Operational Research, Vol. 87, pp.175-187.

Pisinger, D. (1997) *A Minimal Algorithm for the 0-1 Knapsack Problem*, Operations Research, Vol. 45, pp. 758-767.

Plateau, A., Tachat, D. & Tolla, P. (2002) *A Hybrid Search Combining Interior Point Methods and Metaheuristics for 0-1 Programming*, Intl. Trans. in Op. Res., Vol. 9, pp. 731-746.

Prais, M. & Ribeiro, C.C. (2000) *Reactive GRASP: An Application to a Matrix Decomposition Problem in TDMA Traffic Assignment*, INFORMS Journal on Computing, Vol. 12, No. 3, pp. 164-176.

Qingfu, Z., Aimin, Z. & Yaochu, J. (2008) *RM-MEDA: A Regularity Model-Based Multiobjective Estimation of Distribution Algorithm*, IEEE Transactions on Evolutionary Computation, Vol. 12, No. 1, pp. 41-63.

Qiang, L. & Xin, Y. (2005) *Clustering and Learning Gaussian Distribution for Continuous Optimization*, IEEE Transactions on Systems, Man, and Cybernetics-Part C: Applications and Reviews, Vol. 35, No. 2, pp. 195-204.

Rabadi, G., Moraga, R. & Al-Salem, A. (2006) *Heuristics for the Unrelated Parallel Machine Scheduling Problem with Setup Times*, Journal of Intelligent Manufacturing, Vol. 17, pp. 85-97.

Ramos, I.C.O., Goldbarg, M.C., Goldbarg, E.G. & Neto, A.D.D. (2005) *Logistic Regression for Parameter Tuning on an Evolutionary Algorithm*, IEEE, pp. 1061-1068.

Ranjbar, M., Reyck, B.D. & Kianfar, F. (2009) *A Hybrid Scatter Search for the Discrete Time/Resource Trade-Off Problem in Project Scheduling*, European Journal of Operational Research, Vol. 193, pp. 35-48.

Rechenberg, I. (1965) *Cybernetic Solution Path of an Experimental Problem*, Technical Report, Royal Aircraft Establishment Library Translation No. 1112, Farnborough, UK.

Rechenberg, I. (1973) *Evolutionsstrategie: Optimierung Technischer Systeme nach Prinzipien der biologischen Evolution*, Frommann-Holzboog, Stuttgart.

Rechenberg, I. (1994) *Evolutionsstrategie '94*, Frommann-Holzboog, Stuttgart.

Reed, J., Toombs, R. & Barricelli, N.A. (1967) *Simulation of Biological Evolution and Machine Learning: I. Selection of Self-Reproducing Numeric Patterns by Data Processing Machines, Effects of Hereditary Control, Mutation Type and Crossing*, J Theor Biol, Vol. 17, pp. 319-342.

Resende, M.G.C. & Ribeiro, C.C. (2002) *Greedy Randomized Adaptive Search Procedures*, in: Glover, F. & Kochenberger, G. (eds.) State-of-the-Art Handbook of Metaheuristics, Kluwer Academic Publishers, New York, pp. 219-249.

Resende, M.G.C. & Ribeiro, C.C. (2003) *GRASP with Path-Relinking for Private Virtual Circuit Routing*, Networks, Vol. 41, pp. 104-114.

Resende, M.G.C. & Ribeiro, C.C. (2003) *Greedy Randomized Adaptive Search Procedures*, in: Glover, F. & Kochenberger, G.A. (eds.) Handbook of Metaheuristics, Kluwer Academic Publishers, New York.

Resende, M.G.C. & Ribeiro, C.C. (2005) *GRASP with Path-Relinking: Recent Advances and Applications*, in: Ibaraki T., Nonobe, K. & Yagiura, M. (eds.) Metaheuristics: Progress as Real Problem Solvers, Springer, Berlin, pp. 29-63.

Resende, M.G.C., Martí, R., Gallego, M. & Duarte, A. (2010) *GRASP and Path Relinking for the Max-Min Diversity Problem*, Computers & Operations Research, Vol. 37, No. 3, pp. 498-508.

Resende, M.G.C. & Werneck, R.F. (2002) *A GRASP with Path-Relinking for the P-Median Problem*, Technical Report, AT&T Labs Research, Florham Park, NJ.

Ribeiro, C.C & Resende, M.G.C. (2012) *Path-Relinking Intensification Methods for Stochastic Local Search Algorithms*, J Heuristics, Vol. 18, pp. 193-214.

Ribeiro, C.C., Uchoa, E. & Werneck, R.F. (2002) *A Hybrid GRASP with Perturbations for the Steiner Problem in Graphs*, INFORMS Journal on Computing, Vol. 14, pp. 228-246.

Ribeiro, C.C. & Vianna, D.S. (2003) *A Genetic Algorithm for the Phylogeny Problem Using An Optimized Crossover Strategy Based on Path-Relinking*, in: Anais do II Workshop Brasileiro de Bioinformática, Editora Universo, Macaé, pp. 97-102.

Ribeiro, C.C. & Vianna, D.S. (2009) *A Hybrid Genetic Algorithm for the Phylogeny Problem Using Path-Relinking as a Progressive Crossover Strategy*, International Transactions in Operational Research, Vol. 16, No. 5, pp. 641-57.

Ries, J., Beullens, P. & Salt, D. (2012) *Instance-Specific Multi-Objective Parameter Tuning Based on Fuzzy Logic*, European Journal of Operational Research, Vol. 218, pp. 305-315

Rochat, Y. & Taillard, E. (1995) *Probabilistic Diversification and Intensification in Local Search for Vehicle Routing*, Journal of Heuristics, Vol. 1, No. 1, pp. 147-167.

Rosenberg, R.S. (1967) *Simulation of Genetic Populations with Biochemical Properties*, Ph.D. Dissertation, University of Michigan.

Rummery, G.A. & Niranjan, M. (1994) *On-Line Q-Learning Using Connectionist Systems*, Technical Report CUED/F-INFENG-TR 166, Engineering Department, Cambridge University.

Sáez, Y. (2009) *Optimization Using Genetic Algorithms with Micropopulations*, in: Alba, E., Blum, C., Isasi, P., Léon, C. & Gómez, J. A. (eds.) Optimization Techniques for Solving Complex Problems, John Wiley & Sons Inc., New Jersey.

Samuel, L. (1959) *Some Studies in Machine Learning Using the Game of Checkers*, IBM Journal on Research and Development, Vol. 3, pp. 211-229, reprinted in Feigenbaum, E.A. & Feldman J. (eds.) (1963) Computers and Thought, McGraw-Hill, New York, pp. 71-105.

Santana, R., Larrañaga, P. & Lozano, J.A. (2008) *Adaptive Estimation of Distribution Algorithms*, in: Cotta, C., Sevaux, M. & Sörensen, K. (eds.) Adaptive and Multilevel Metaheuristics, Springer, Berlin, pp. 177-197.

Sagarna, R. & Lozano, J. (2005) *On the Performance of Estimation of Distribution Algorithms Applied to Software Testing*, Appl Artif Intell. Vol, 19, No. 5, pp. 457-89.

Santana, R. (2005) *Estimation of Distribution Algorithms with Kikuchi Approximations*, Evolutionary Computation, Vol. 13, No. 1, pp. 67-97.

SAS Institute Inc (2007) *JMP, Design of Experiments Guide*, SAS, Cary, North Carolina.

Santana, R., Larrañaga, P. & Lozano, J.A. (2008) *Combining Variable Neighborhood Search and Estimation of Distribution Algorithms in the Protein Side Chain Placement Problem*, Journal of Heuristics, Vol. 14, No. 5, pp. 519-547.

Sarin, S., Karwan, M. & Rardin, R. (1988) *Surrogate Duality in a Branch-And-Bound Procedure for Integer Programming*, European Journal of Operational Research, Vol. 33, pp. 326-333.

Sastry, K., Pelikan, M. & Goldberg, D.E. (2006) *Efficiency Enhancement of Estimation of Distribution Algorithms*, in: Pelikan, M., Sastry, K. & Cantú-Paz, E. (eds.) Scalable Optimization via Probabilistic Modeling: From Algorithms to Applications, Springer, New York, pp. 161-185.

Schwefel, H.-P. (1965) *Kybernetische Evolution als Strategie der Experimentellen Forschung in der Strömungstechnik*, Technical Report, Diplomarbeit Hermann Fottinger Institut für Strömungstechnik, Technische Universität, Berlin, Germany.

Schwefel, H.-P. (1974) *Adaptive Mechanismen in der Biologischen Evolution und ihr Einflus auf die Evolutionsgeschwindigkeit*, Interner Bericht der Arbeitsgruppe Bionik und Evolutionstechnik am Institut fnr Mess- und Regelungstechnik, TU Berlin.

Shakya, S. & McCall, J. (2007) *Optimization by Estimation of Distribution with DEUM Framework Based on Markov Random Fields*, International Journal of Automation and Computing, Vol 4, No. 3, pp. 262-272.

Shakya, S. & Santana, R. (2008) *An EDA Based on Local Markov Property and Gibbs Sampling*, Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation, GECCO'08, ACM, New York, NY, pp. 475-476.

Senyu, S. & Toyada, Y. (1967) *An Approach to Linear Programming with 0-1 Variables*, Management Science, Vol. 15, pp. B196-B207.

Shih, W. (1979) *A Branch & Bound Method for the Multi-Constraint Zero-One Knapsack Problem*, Journal of Operational Research Society, Vol. 30, pp. 369-378.

Sikora, R.T. (2008) *Meta-Learning Optimal Parameter Values in Non-Stationary Environments*, Knowledge-Based Systems, Vol. 21, pp. 800–806.

Silver, E. (2004) *An Overview of Heuristic Solution Methods*, Journal of the Operational Research Society, Vol. 55, pp. 936-956.

Sima, U. and Gülsen, E. (2005) *Improvements to Penalty Based Evolutionary Algorithms for the Multidimensional Knapsack Problem Using A Gene-Based Adaptive Mutation Approach*, GECCO 2005, pp. 1257-1264.

Soares, C., Brazdil, P. and Kuba, P. (2004) *A Meta-Learning Approach to Select the Kernel Width in Support Vector Regression*, Machine Learning 54 (3), pp. 195-209.

Su, Z.-P., Jiang, J.-G., Liang, C. Y. and Zhang, G. F. (2011) *Path Selection in Disaster Response Management Based on Q-Learning*, International Journal of Automation & Computing 8(1), pp. 100-106.

Sutton, R.S. (1984) *Temporal Credit Assignment in Reinforcement Learning*, Unpublished Ph.D. Dissertation, University of Massachusetts, Amherst.

Sutton, R.S. (1988) *Learning to Predict by the Method of Temporal Differences*, Machine Learning, Vol. 3, pp. 9-44.

Sutton, R.S. (1996) *Generalization in Reinforcement Learning: Successful Examples Using Sparse Coarse Coding*, in Touretzky, D. S., Mozer, M. C. & Hasselmo, M. E. (eds.) Advances in Neural Information Pocessing Systems, Proceedings of the 1995 Conference, MIT Press, Cambridge, MA, pp. 1038-1044.

Sutton, R.S. & Barto, A. G. (1998) *Reinforcement Learning: An Introduction*, MIT Press, Cambridge, MA.

Taillard, E.D., Gambardella, L.M., Gendreau, M. & Potvin, J.Y. (2001) *Adaptive Memory Programming, A Unifed View of Metaheuristics*, European Journal of Operational Research, Vol. 135, pp. 1-16.

Talbi, E.G. (2009) *Metaheuristics, From Design To Implementation*, University of Lille, John Wiley & Sons, Inc., New Jersey.

Tamilarasi, A. (2010) *Tunings of Parameters and Pheromone Update Strategy in Ant Colony Optimization*, Journal of Advanced Manufacturing Systems, Vol. 9, No. 1, pp. 73-83.

Thesen, A. (1973) *Scheduling of Computer Programs in a Multiprogramming Environment*, BIT, Vol. 13, pp. 208-216.

Tesauro, G. & Kephart, J.O. (2002) *Pricing in Agent Economies Using Multi-Agent Q-Learning*, Autonomous Agents and Multi-Agent Systems, Vol. 5, pp. 289-304.

Torre, A.L., Pe˜na, J.-M., Muelasa, S. & Freitas, A.A. (2010) *Learning Hybridization Strategies in Evolutionary Algorithms*, Intelligent Data Analysis, Vol. 14, pp. 333-354.

Tsitsiklis, J.N. (1994) *Asynchronous Stochastic Approximation and Q-Learning*, Machine Learning, Vol. 16, pp. 185-202.

Tsutsui, S. (2002) *Probabilistic Model-Building Genetic Algorithms in Permutation Representation Domain Using Edge Histogram*, Proceedings of the 7th Int. Conf. on Parallel Problem Solving from Nature, PPSN VII, Springer-Velag, pp. 224-233

Usberti, F.L., França, P.M. & França, A.L.M. (2011) *GRASP with Evolutionary Path-Relinking for the Capacitated Arc Routing Problem*, Computers and Operations Research, doi:10.1016/j.cor.2011.10.014.

Uwe, A. & Jingpeng, L. (2007) *An Estimation of Distribution Algorithm for Nurse Scheduling*, Ann Oper Res., Vol. 155, pp. 289-309.

Valasek, J., Doebbler, J., Tandale, M.D. & Meade, A.J. (2008) *Improved Adaptive-Reinforcement Learning Control for Morphing Unmanned Air Vehicles*, IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics, Vol. 38, No. 4, pp. 1014-1020.

Vallada, E. & Ruiz, R. (2010) *Genetic Algorithms with Path Relinking for the Minimum Tardiness Permutation Flowshop Problem*, Omega, Vol. 38, No. 1-2, pp.57-67.

Vasquez, M. & Hao, J.K. (2001) *A Logic-Constrained Knapsack Formulation and a Tabu Algorithm for the Daily Photograph Scheduling of An Earth Observation Satellite*, Comput. Optim. Appl., Vol. 20, pp. 137-157.

Vasquez, M. & Hao, J.K. (2001) *Une Approche Hybride Pour Le Sac à Dos Multidimensionnel En Variables 0-1*, RAIRO Operations Research, Vol. 35, pp. 415-438.

Vasquez, M. & Vimont, Y. (2005) *Improved Results on the 0-1 Multidimensional Knapsack Problem*, European Journal of Operational Research, Vol. 165, pp. 70-81.

Villegas, J.G., Prins, C., Prodhon, C., Medaglia, A.L. & Velasco, N. (2011) *A Grasp with Evolutionary Path Relinking for the Truck and Trailer Routing Problem*, Computers & Operations Research, Vol. 38, pp. 1319-1334.

Vimont, Y., Boussier, S. & Vasquez, M. (2008) *Reduced Costs Propagation in an Efficient Implicit Enumeration for the 01 Multidimensional Knapsack Problem*, J. Comb. Optim., Vol. 15, pp. 165-178.

Vogt, L., Poojari, C.A. & Beasley, J.E. (2007) *A Tabu Search Algorithm for the Single Vehicle Routing Allocation Problem*, Journal of the Operational Research Society, Vol. 58, pp. 467-480.

Volgenant, A. & Zoon, J.A. (1990) *An Improved Heuristic for the Multidimensional 0-1 Knapsack Problems*, Journal of the Operational Research Society, Vol. 41, pp. 963-970.

Wang, X. & Tang, L. (2009) *A Population-Based Variable Neighborhood Search for the Single Machine Total Weighted Tardiness Problem*, Computers & Operations Research, Vol. 36, pp. 2105-2110.

Wanga, Y.-C. & Ushera, J.M. (2004) *Learning Policies for Single Machine Job Dispatching*, Robotics and Computer-Integrated Manufacturing, Vol. 20, pp. 553-562.

Wang, B., Wang, J., Song, X & Liu, F. (2009) *Q-Learning-Based Adaptive Waveform Selection in Cognitive Radar*, Int. J. Communications, Network and System Sciences, Vol. 7, pp. 669-674.

Wang, J., Kuang, Z., Xu, X. & Zhou, Y. (2009) *Discrete Particle Swarm Optimization Based on Estimation of Distribution for Polygonal Approximation Problems*, Expert Systems with Applications, Vol. 36, pp. 9398-9408.

Wang, L. & Fang, C. (2012) *An Effective Estimation of Distribution Algorithm for the Multi-Mode Resource-Constrained Project Scheduling Problem*, Computers & Operations Research, Vol. 39, pp. 449-460.

Wang, Y. & Silva, C.W. (2010) *Sequential Q-Learning with Kalman Filtering for Multirobot Cooperative Transportation*, IEEE/ASME Transactions on Mechatronics, Vol. 15, No. 2, pp. 261-268.

Wassan, N. (2006) *A Reactive Tabu Search For The Vehicle Routing Problem*, Journal of the Operational Research Society, Vol. 57, pp. 111-116.

Wassan, N. (2007) *Reactive Tabu Adaptive Memory Programming Search for the Vehicle Routing Problem With Backhauls*, Journal of the Operational Research Society, Vol. 58, pp. 1630-1641.

Watkins, C.J.C.H. (1989) *Learning from Delayed Reward*, Ph.D. Dissertation, Cambridge University.

Watkins, C.J.C.H. & Dayan, P. (1992) *Q-Learning*, Machine Learning, Vol. 8, pp. 279-292.

Webster`s New Universal Unbridged Dictionary (1996) Random House Value Publishing, Inc., New York, NY.

Weinberg, R. (1970) *Computer Simulation of a Living Cell*, Ph.D. Dissertation, University of Michigan.

Weingartner, H.M. & Ness, D.N. (1967) *Methods for the Solution of the Multi-Dimensional 0/1 Knapsack Problem*, Operations Research, Vol., pp. 83-103.

Wiering, M, Schmidhuber, J. & Elvezia, I.C. (1998) *Fast Online Q(λ)*, Machine Learnin, Vol. 33, pp. 105-115.

Wilbaut, C. & Hanafi, S. (2008) *A Survey of Effective Heuristics and their Application to a Variety of Knapsack Problems*, IMA Journal of Management Mathematics, Vol. 19, pp. 227-244.

Wilbaut, C. & Hanafi, S. (2009) *New Convergent Heuristics for 0–1 Mixed Integer Programming*, European Journal of Operational Research, Vol. 195, pp. 62-74.

Wilbaut, C., Hanafi, S. & Salhi, S. (2008) *A Survey of Effective Heuristics and their Application to a Variety of Knapsack Problems*, IMA Journal of Management Mathematics, Vol. 19, pp. 227-244.

Wilbaut, C., Salhi, S. & Hanafi, S. (2009) *An Iterative Variable-Based Fixation Heuristic for the 0-1 Multidimensional Knapsack Problem*, European Journal of Operational Research, Vol. 199, pp. 339-348.

Wolpert, D.H. & Macready, W.G. (1997) No Free Lunch Theorems for Optimization, IEEE Transactions on Evolutionary Computation, Vol, 1, No. 1, pp. 67-82.

Wong, K.Y.K. (2008) Parameter Tuning for Ant Colony Optimization: A Review, Proceedings of the International Conference on Computer and Communication Engineering, May 13-15, Kuala Lumpur, Malaysia.

Xiao, J., Yan, Y. & Zhang, J. (2009) *HPBIL: A Histogram-Based EDA for Continuous Optimization*, Applied Mathematics and Computation, Vol. 215, No. 3, pp. 973-982.

Yang, X.-S. (2010) *Engineering Optimization: An Introduction with Metaheuristic Applications*, John Wiley & Sons, Inc., Hooboken, New Jersey.

Yao, X. & Liu, Y. (2005) *Machine Learning*, in: Burke, E.K. & Kendall, G. (eds.) Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques, Springer, New York, NY.

Yin, P.-Y., Glover, F., Laguna, M. & Zhu, J.-X. (2010) *Cyber Swarm Algorithms - Improving Particle Swarm Optimization Using Adaptive Memory Strategies*, European Journal of Operational Research, Vol. 201, pp. 377-389.

Yuan, B., Orlowska, M. & Sadiq, S. (2008) *Extending a Class of Continuous Estimation of Distribution Algorithms to Dynamic Problems*, Optimization Letters, Vol. 2, pp. 433-443.

Zhang, G.Q. & Lai, K.K. (2006) *Combining Path Relinking and Genetic Algorithms for the Multiple-Level Warehouse Layout Problem*, European Journal of Operational Research, Vol. 169, pp. 413-425.

Zhang, Q., Sun, J., Tsang, E. & Ford, J. (2003) *Combination of Guided Local Search and Estimation of Distribution Algorithm for Solving Quadratic Assignment Problem*, Proceedings of the Bird of a Feather Workshops, Genetic and Evolutionary Computation Conference, pp. 42-48.

Zhang, Q., Sun, J., Tsang, E. & Ford, J. (2006) *Estimation of Distribution Algorithm with 2-Opt Local Search for the Quadratic Assignment Problem*, in: Lozano J.A., Larranaga P. & Inza I. (eds.) Towards a New Evolutionary Computation, Advances in the Estimation of Distribution Algorithms, Springer-Verlag, Berlin Heidelberg.

Zhang, Q., Zhou, A. & Jin, Y. (2008) *RM-MEDA: A Regularity Model-Based Multiobjective Estimation of Distribution Algorithm*, IEEE Transactions on Evolutionary Computation, Vol. 12, No. 1, pp. 41-63.

Zhang, Y. & Bhattacharyyaz, S. (2007) *Effectiveness of Q-Learning as a Tool For Calibrating Agent-Based Supply Network Models*, Enterprise Information Systems, Vol. 1, No. 2, pp. 217-233.

Zhang, Z., Zheng, L. & Weng, M.X. (2007) *Dynamic Parallel Machine Scheduling with Mean Weighted Tardiness Objective by Q-Learning*, Int J Adv Manuf Technol, Vol. 34, pp. 968-980.

Zhou, Y. & Wang, J. (2010) *Neural Network Combined with Estimation of Distribution for Max-Cut Problem*, ICIC Express Letters, Vol. 4, No. 4, pp. 1161-1166

Zhong, X., Ding, J., Li, W. & Zhang, Y. (2008) *Robust Airfoil Optimization with Multi-Objective Estimation of Distribution Algorithm*, Chinese Journal of Aeronautics, Vol. 21, No. 4, pp. 289-295.

VITA

Arif Arin

Engineering Management and Systems Engineering Department

Norfolk, VA 23529

Arif Arin received his Bachelor's Degree in Industrial Engineering from the Turkish Air Force Academy, Turkey in 1995 and his Master of Science Degree in Systems Engineering from Air Force Institute of Technology (AFIT), USA, in 2002. He taught an Operations Research course at Izmir Economy University, and a Computer Technologies course at the Turkish Air Force Noncommissioned Officer Vocational College, Turkey. He is an IEEE member. His research interests include optimization methods, metaheuristics, machine learning, scheduling and design of experiments.