Université de Tunis
Institut Supérieur de Gestion
Laboratoire LARODEC

Mémoire de Mastère en Modélisation

The Optimal Stopping Knapsack Problem

Elaboré par:

Hajer Ben Romdhane

Supervisé par: **Dr. Saoussen Krichen**

Année Universitaire 2009-2010

To my Dear Parents,

all my Family and my Friends.

Acknowledgements

First and above all, I praise God, the almighty for providing me this opportunity and granting me the capability to proceed successfully.

This work appears in its current form due to the assistance and guidance of several people. I would therefore like to take this opportunity to convey my sincere thanks to all of them.

I am deeply indebted to my supervisor Dr. Saoussen Krichen for her continuous supervision, wise advice, and support. I greatly benefited from her guidance during this work.

My sincere thanks go to all the people who have contributed in this search, by information, advice, criticism or encouragement.

Contents

Glossary **8**

Abstract **10**

General Introduction **11**

Knapsack and Optimal Stopping Problems: A review **14**

1 Knapsack Problems **14**

1.1 Introduction 14

1.2 Notation 16

1.3 Statement of the problem 16

1.4 Variants and extensions of the knapsack problem 17

1.4.1 Static knapsack problems 20

1.4.2 Dynamic knapsack problems 30

1.5 General comments 39

1.6 Conclusion 40

2 Optimal Stopping Problems **42**

2.1 Introduction 42

2.2 Notation 44

2.3 Statement of the problem 44
2.4 Variants and extensions of the optimal stopping problem 47
2.5 Conclusion 59

The Optimal Stopping Knapsack Problem 62

3 The Optimal Stopping Knapsack Problem 62
3.1 Introduction 62
3.2 Statement of the problem 64
3.3 Notation 65
3.4 Mathematical formulation 66
3.5 The proposed solution approach 70
3.6 Conclusion 79

4 Computational Experiments 80
4.1 Introduction 80
4.2 The experimental settings 81
4.3 Interpretation of the results 82
4.3.1 Small instances 82
4.3.2 Large instances 86
4.4 Conclusion 91

General Conclusion 92

Bibliography 97

List of Figures

2.1 Chart of the OSP 46

3.1 Decision Process of the OSKP 65
3.2 Chart of the OSKP 72
3.3 A decision strategy example for $n = 5$ 78

4.1 Progress of the number of possible collections of items in terms of problem sizes 84
4.2 Effectiveness of the DSA results compared by those of a static algorithm 84
4.3 Evaluation of the percentage of filling of the knapsack for different problem sizes 85
4.4 The CPU time behavior in terms of problem sizes 86
4.5 The CPU time progress for large n 88
4.6 The effectiveness in filling the knapsack for both utility functions . . 88
4.7 Variation of the first load stage by comparing U_1 and U_2 89
4.8 The LBLS behavior in terms of U_1 and U_2 compared to the static case 90

List of Tables

1.1 Some Hybrid Knapsack Problems 19
1.2 Summary of Literature Review on Static KPs 29
1.3 Summary of Literature Review on Dynamic KPs 38
1.4 Comparison Table of Static and Dynamic KPs 40

2.1 Summary of some OSP variants 59

3.1 Comparison of utility functions values 69
3.2 Example with $n = 5$ 74
3.3 Expected utilities for $n = 5$ 74
3.4 Expected utilities for $n = 5$ 77

4.1 Experimental results for the DSA 83
4.2 Overall results for the OSA 87

List of Algorithms

1 The Fixed Choice Online Algorithm $\mathbf{G}(\overline{\mathbf{d}})$ 31
2 Approximative KP Algorithm 33
3 The Optimal Solution Algorithm of the OSKP 73
4 The Decision Strategy Algorithm of the OSKP 76

Glossary

1RB	One out of the r Best
AR	Average Reward
B&B	Branch and Bound
BKP	Bounded Knapsack Problem
BOSP	Bilateral Optimal Stopping Problem
DCKP	Disjunctively Constrained Knapsack Problem
DM	Decision Maker
DP	Dynamic Programming
DSA	Decision Strategy Algorithm
DSKP	Dynamic and Stochastic Knapsack Problem
FLS	First Load Stage
FPTAS	Fully Polynomial-Time Approximation Scheme
GA	Genetic Algorithm
GIP	Group Interviewed Problem
KAP	Knapsack Auction Problem
KP	Knapsack Problem
KP01	Binary Knapsack Problem
KSP	Knapsack Secretary Problem
LBLS	The percentage of Loading Before the Last Stage

MASP	Multi-Attribute Secretary Problem
MinKP	Minimization Knapsack Problem
MOEA	Multi-Objective Evolutionary Algorithms
MOKP	Multi-Objective 0-1 Knapsack Problem
NLI	Number of Loaded Items
NPC	Number of Possible Collections
OKP	Online Knapsack Problem
OSA	Optimal Solution Algorithm
OSKP	Optimal Stopping Knapsack Problem
OSP	Optimal Stopping Problem
PF	Percentage of Filling
PTAS	Polynomial-Time Approximation Scheme
SKP	Stochastic Knapsack Problem
SP	Secretary Problem
SSP	Subset Sum Problem
UKP	Unbounded Knapsack Problem
WSP	Weighted Secretary Problem

Abstract

The Optimal Stopping Knapsack Problem is defined as follows. A sequence of items arriving over time, one at a time, without any prior information. Each item is characterized by a specific weight and an associated reward. Once arrived, an item is evaluated to decide whether to select it immediately or to delay the decision to a next stage. The main objective is to select the best collection of items, that maximizes the overall reward under the capacity constraint. The selection process can be stopped at any time, even before observing all items, if the capacity constraint is exhausted. We propose a solution approach that decomposes the original problem dynamically and incorporates an optimal stopping rule in order to decide whether to load or not the currently arriving items. We illustrate the proposed approach by a numerical experimentation and analyze the generated results.

General Introduction

In business world, decision makers (DM) are required to make relevant decisions for daily opportunities. Typically, good opportunities are rare, and once encountered, we should take advantage of it. Therefore, good decisions have to consider outcomes as well as deadlines. Making decisions became difficult as the number of choices grows while the budget is limited. In such a case, we are in front of a sequential resource allocation problem. In this search, we deal with the knapsack problem.

The knapsack problem is one of the widely and extensively studied resource allocation problem. In its basic version, we are given a number of items from which we are required to select a subset to carry it in a fixed capacity knapsack. Items differ by their value and their required place in the knapsack. The aim is to load items which maximize the overall reward without exceeding the capacity.
However, it is unusual to receive offers at the same time. Generally, offers appear sequentially over time. Thus, it is more realistic to consider the dynamic knapsack problem, in which we receive offers over periods. The major difficulty in such problem is the fact that we have no information about incoming offers, and we have to decide on the current offer before observing the next one. The dynamic knapsack problem was introduced by Marchetti-Spaccamela and Vercellis in 1995, and studied since by few researchers.

In this search, we are concerned by a dynamic version of the knapsack, that allows to delay the decision about incoming items for next stages. To the best of our knowledge, no works was concerned before by the dynamic knapsack with delay. Indeed, incoming offers are available for a certain time, so it is convenient to reconsider it for a number of periods. That is, if an item is received in the current period, the DM can postpone the decision about the item for a next stage. However, if selected, the item cannot be taken out the knapsack. Therefore, the problem asks to the opportune time to select a specific offer.

To solve this problem, we may appeal to the optimal stopping terminology. An optimal stopping problem arises when we are in front of a sequence of offers, from which we are asked to select the best one. Offers are presented to the observer one by one, without any prior information about the following ones. Once an offer is received, we have to decide immediately whether to accept it and quit the process, or discard it to evaluate the next one. Note that the decision is irrevocable.

Thus, our problem can be viewed as a multiple-choice optimal stopping problem with delay, in which we are allowed to select more than a unique offer and to reconsider offers not selected from previous stages. Recognizing these similarities, we propose a solution approach managed by an optimal stopping rule. Therefore, our purpose is to derive a stopping strategy, to decide which offer to discard and on which we shall stop the process. We develop a dynamic formulation for our problem, namely the optimal stopping knapsack problem.

The present report is organized as follows. We review in the first part the existing literature for both the knapsack and the optimal stopping problems. This will enable us to situate our contribution among existing works. The second part will be devoted to explain our contribution. We discuss in a first part the problem and the proposed method. We illustrate in the second chapter by a numerical investigation to measure the effectiveness of the presented algorithm.

Knapsack and Optimal Stopping Problems: A review

Chapter 1

Knapsack Problems

1.1 Introduction

The knapsack problem (KP) is a well known and widely studied combinatorial optimization problem. Researchers are interested in the study of KPs due to its theoretical and practical importance, Martello and Toth (1990). Theoretically, it has a simple structure and the ability to solve more complex optimization problems. And practically, the KP models a wide range of industrial situations belonging to the domains of transportation as cargo loading (Bellman and Dreyfus, 1962), cutting stock (Gilmore and Gomory, 1966), telecommunication, reliability, advertisement, budget allocation, financial management (Cord, 1964; Kaplan, 1966; Hansmann, 1961) and cryptography (Diffe and Hellman, 1976).

The KP derives its name from the hiker's problem of selecting which items, among a given set of items, to fill his knapsack in such a way that the overall value of selected items is maximized such that the knapsack weight capacity is not exceeded.

KP has seen several variations over the years; the difference lies in the items and resources distribution, the objectives considered, etc. In its binary form, an item

is either taken or left while in the fractional form, the DM can load only a fraction of the item. In the bounded KP there are a number of copies from each item type, and an unlimited number of copies in the unbounded form. In the case of the multiple-choice KP, items are classed by group while in the multidimensional and the multiple forms, a number of knapsacks are supposed to be filled. If the weight and the value of each item vary depending on the knapsack it's assigned to, the problem is called multi-objective 0-1 KP.
Many hybrid forms of the KP have been studied also over years such as: the knapsack auction problem, the stochastic KP, the online KP and the dynamic and stochastic KP.
Researchers have considered other type of objectives in studying the KP: the subset sum problem is a special case of the KP, where the objective is to load the weightiest collection of items without exceeding the knapsack capacity. Besides, many maximization KPs could be transformed to its equivalent minimization version while the objective is to minimize the profit of the not packed items.
KPs are applied either as stand-alone problems, where a general integer programming problem is transformed into a KP (Salkin, 1975; Sysloet al., 1983), or as subproblems of more complex programming models (Syslo et al., 1983; Balas, 1975).

They belong to NP-hard problems (Garey and Johnson, 1979). Many solution procedures have been proposed to solve problems from the KP family, including exact and heuristic algorithms.

Throughout this chapter, we investigate the KP. First we present the notations followed by the problem description. In the fourth section, we are going to detail some variants of the KP and to present their associated formulations. Section 1.5 deals with a comparative study between discussed versions of KPs.

1.2 Notation

Following are the notations adopted in this chapter.

Symbols	***Explanations***
c	Weight capacity of the knapsack
c_j	Weight capacity of knapsack j
n	Number of items
m	Number of knapsacks
r	A deterministic return threshold
b_j	Number of items of type j
w_i	Weight of the item i
v_i	Value of the item i
d_i	Profit density of the item i
$\overline{d}$	Threshold density
$x_i \in \{0,1\}, i = \{1,...,n\}$	If item i is loaded $x_i = 1$, otherwise $x_i = 0$
$x_{ij} \in \{0,1\}$, $i = \{1,...,n\}$,	If item i is assigned to knapsack j, $x_i = 1$,
$j = \{1,...,m\}$	otherwise $x_{ij} = 0$
$Z(x)$	The objective function

1.3 Statement of the problem

A KP holds when we are asked to select a number of items, from a given set, to be carried in a knapsack. Each item is associated with value v_i and weight w_i, and the knapsack has a limited capacity c. The KP is to fill the knapsack in such a way that its content has maximum value under the capacity constraint.

The problem is stated as:

$$\begin{aligned} \text{Maximize} \quad & Z(x) = \sum_{i=1}^{n} v_i x_i \\ \text{Subject to} \quad & \sum_{i=1}^{n} w_i x_i \leq c, \\ & x_i = 0 \text{ or } 1; \qquad i = 1, 2, ..., n \end{aligned} \tag{1.1}$$

The above description corresponds to the *0-1 KP* (or *Binary KP*). Several variants of the basic problem were studied in the literature. We dedicate the following section to examine variants of the problem.

1.4 Variants and extensions of the knapsack problem

The KP knew several extensions and variations over years. Whatsoever the objective function type, items features distribution, the number of objectives considered or the number of knapsacks to be filled, the knapsack family of problems asks for beneficial ways to fill a specific amount of resource.

Nevertheless, we might distinguish between two fundamental aspects: static and dynamic problems. In static problems, a stream of items are present simultaneously for the selection and items types are known a priori. However, dynamic KPs suppose a sequence of alternatives arriving sequentially over time, without any prior information, so that items are packed into the knapsack one by one.

In what follows, we will investigate some static and dynamic variants of the KP that are mainly related to our contribution. However, a wider list of variants exists, from which we present a brief description for hybrid versions in Table 1.1.

Some Definitions

- *ε-Approximate Algorithm*

An ε-approximate algorithm is an algorithm that yields a solution with a relative deviation from the optimum of less than the approximation ratio ε, Guntzer and Jungnickel (2000).

- *Competitive Algorithm*

A competitive algorithm for an online problem has the property that its performance on any sequence of requests is within a constant factor of the performance of any other algorithm on the same sequence, Manasse et al. (1988).

Table 1.1: Some Hybrid Knapsack Problems

KP Variants	Description	Researchers	Application Domains
Collapsing KP	Non constant capacity knapsack; the knapsack size is a non-increasing function of the number of items loaded	Guignard and Posner (1978) Fayard and Plateau (1994) Pferschy et al. (1997) Jigang and Srikanthan (2006)	• Satellite communications • Computer operation in a time sharing environment
Quadratic KP	Maximizing a quadratic objective function subject to a linear capacity constraint	Gallo et al. (1980) Martello and Toth (1990) Pisinger et al. (1998, 2004, 2007)	• Optimal sites for communication satellite earth stations • The location of railway stations
Non-linear KP	A nonlinear optimization problem with one constraint	Hochbaum (1995) Bretthauer and Shetty (2002) Elhedhli (2005)	• Production and inventory management
Nested KP	Nesting is that some knapsacks are allowed to be subsets of other ones	Armstrong et al. (1982) Johnston and Khan (1995)	• Catalog planning • Sales resource allocation
K. Sharing P.	Maximizing the minimal value of a number of linear functions subject to a single linear constraint	Brown (1979) Yamada et al. (1998) Fujimoto and Yamada (2006) Belgacem and Hifi (2007)	• Project selection
Inverse-Parametric KP	A KP with a parametric costs and a value function $v(t)$. For a given target v^*, the problem is to determine the minimum value of the parameter t	Burkard and Pferschy (1995)	• Real estate marketing
KP with Setups	Select a number of items, belonging to the same family of items, given that an item can only be selected if a setup charge for placing the family of items in the knapsack is incurred	Michel et al. (2008) Altay et al. (2008)	• Scheduling problems

1.4.1 Static knapsack problems

The static aspect of KPs is defined as: given a collection of items, each with a certain value and weight, and a limited capacity knapsack, which items should be packed in the knapsack in order to maximize contents value. The common characteristics of static KPs are:

- items available at the same time
- items values and weights are known beforehand

However, other objectives and specifications were considered, giving rise to the appearance of several variants of the problem. We focus in the rest of this section on the most important.
We begin by explaining the binary form of the KP. Then we consider a generalization; the bounded KP, followed by a special case; the subset sum problem. After that we study the multi-objective KP, an extension of the binary form, as well as the minimization version. Finally we present the stochastic variant. Table 1.1 illustrates resolution methods and applications of studied variants.

❶ The 0-1 Knapsack Problem (KP01)

Problem aspect	Static and deterministic
Available information	Items features

The binary KP is the basic form of KPs. The KP01 is defined as: given a set of n items, with different sizes and values, and a knapsack with a finite capacity c, the objective is to select a collection of items which maximize the total value without exceeding the knapsack capacity. The problem formulation is given in (1.1).
Bellman was the first to propose an exact algorithm to solve the KP01, based on the dynamic programming (DP) in the fifties. Dantzig (1957) produced the first upper

bound using a method to resolve the solution of the continuous relaxation of the problem. In 1967, Kolesar gave the first branch and bound (B&B) algorithm. Many other algorithms were presented over the next years based on exact and heuristic methods, from which we mention the most famous: Martello and Toth (1980) and Pisinger (1997) have proposed algorithms based on B&B, Fayad and Plateau (1975), Pisinger (1995) and Martello and Toth (1997) proposed DP algorithms. Hybrid algorithms combining B&B and DP were also proposed by Plateau and Elkihel (1985) and Martello and Toth (1984).

❷ The Bounded Knapsack Problem (BKP)

Problem aspect	Static and deterministic
Available information	Items features

The BKP is a generalization of the KP01 where from each item type there are a single copy, $b_j = 1$ for all $j \in n$. In the BKP, from each item type there are up to b_j items of type j that can be put in the knapsack.
The problem is then stated as:

$$\begin{aligned} \text{Maximize} \quad & Z(x) = \sum_{i=1}^{n} v_i x_i \\ \text{Subject to} \quad & \sum_{i=1}^{n} w_i x_i \leq c, \\ & x_i \in \{0, 1, 2, ..., b_i\}, \quad i = 1, 2, ..., n \end{aligned} \tag{1.2}$$

The BKP is closely related to the KP01, that's why all mathematical and algorithmic approaches of the KP01 could be extended to solve the BKP, and BKPs could be

converted to the binary model, Martello and Toth (1990). Therefore, only some algorithms were specially developed for the BKP. We quote as example Martello and Toth (1977), Ingargiola and Korsh (1977), Bulfin et al. (1979). Martello and Toth (1990) demonstrate that their algorithm MT2, which is applied after conversion of the problem at its binary equivalent, is more effective. Recently, Pferschy (1999) developed an improved DP algorithm and Arie Tamir (2009) developed new bounds for solving the BKP.

The bounded version of the KP can be extended to the unbounded KP (UKP), where an unlimited number of copies of each item type is available. DP and B&B were used for solving exactly the UKP. Cargo loading and packing are examples of UKP industrial applications (Bellman and Dreyfus, 1962).

❸ The Subset Sum Problem (SSP)

Problem aspect	Static and deterministic
Available information	Items features

The SSP is a special case of the KP, arising when the weight of an item is equal to its reward. The objective is to select the subset of items, with the largest weight, which still fit in the knapsack. Formally, given n items, each with an associated weight w_i, and a knapsack with a finite capacity c, find the collection of items whose total weight combination is equal or closest to c.

The problem can be stated as:

$$\begin{aligned} \text{Maximize} \quad & Z(x) = \sum_{i=1}^{n} w_i x_i \\ \text{Subject to} \quad & \sum_{i=1}^{n} w_i x_i \leq c, \\ & x_i = 0 \text{ or } 1; \qquad i = 1, 2, ..., n \end{aligned} \tag{1.3}$$

The SSP is known to be NP-hard and it can be solved in a pseudo-polynomial time (Garey and Johnson, 1979). The SSP can be solved simply by any method of the KP01; this implies some specific treatments, Martello and Toth (1990). Several exact and heuristic algorithms were proposed for the SSP since the fifties. The first algorithm to solve exactly the problem was the MTSL of Martello and Toth (1984), combining DP and tree-search. Pisinger (1999) and Soma and Toth (2002) have proposed improved algorithms, having a polynomial performance equivalent to that of the MTSL, but with better worst case behavior Soma and Toth (2002). Approximate approaches were also proposed by a number of researchers as Ghosh and Chakravarti (1999), which used a local search heuristic, and Wang (2003) have presented an improved genetic algorithm (GA).

❹ The Multi-Objective 0-1 Knapsack Problem (MOKP)

Problem aspect	Static and deterministic
Available information	Items features

In the MOKP, a binary KP is considered with m knapsacks of capacities $c_1, c_2, ..., c_m$ (i.e., m objectives and m constraints). The profit and weight of each item are vary-

ing according to which knapsack the item is placed; v_{ij} and w_{ij} are respectively the profit and the weight associated with assigning item i to knapsack j.
We formulate the problem as:

$$\begin{array}{lll} \text{Maximize} & Z(x) = (z_1(x), z_2(x), ..., z_m(x)) & \\ \text{Where} & z_i(x) = \sum_{j=1}^{n} v_{ij} x_j & \\ \text{Subject to} & \sum_{j=1}^{n} w_{ij} x_j \leq c_j, & j = 1, 2, ..., m, \\ & x_{ij} = 0 \text{ or } 1, & \text{for all } i \text{ and } j \end{array} \tag{1.4}$$

This problem arises often in the problem of assigning jobs to several machines, in loading problems, etc. DP algorithms were proposed for the MOKP to derive the efficient set (Klamroth and Wiecek, 2000; Bazgan et al., 2009), as well as a two phases method including a B&B algorithm (Visée et al., 1998). Captivo et al. (2003) reduced the problem to a bi-objective shortest path problem solved by a labeling algorithm, Bazgan et al. (2009).

Several multi-objective evolutionary algorithms (MOEA) have been proposed to solve the MOKP because of their suitability in solving multi-objective optimization problems, if compared with classical methods. Zitzler and Thiele have suggested two MOEAs; SPEA (1999) and its improved version SPEA2 (2001). Knowles and Core (2000) have proposed the M-PAES; an hybrid MOEA, and recently Alves and Almeida (2007) have proposed a more specific MOEA called MOTGA.

Foued et al. (2000) proposed two metaheuristics. The first one based on GA, while the second on tabu search.

❺ The Disjunctively Constrained Knapsack Problem (DCKP)

Problem aspect	Static and deterministic
Available information	Items features

The DCKP is an extension of the 01KP. In addition to the classical description, the DCKP should satisfy a disjunctive constraint. The additional constraint suppose that there are incompatible pairs of items in the input set. The DCKP is then to carry the subset of items maximizing the profit without exceeding the knapsack capacity, nor packing together two items from a conflict pair.
We denote by E the set of incompatible pairs such that $E \subseteq \{(i,j)|1 \leq i \neq j \leq n\}$. If $(i,j) \in E$, items i and j are not allowed to be loaded in the same knapsack.
Given this description, the problem is formulated as follows:

$$\begin{aligned} \text{Maximize} \quad & Z(x) = \sum_{i=1}^{n} v_i x_i \\ \text{Subject to} \quad & \sum_{i=1}^{n} w_i x_i \leq c, \\ & x_i + x_j \leq 1, \qquad \forall (i,j) \in E \\ & x_i = 0 \text{ or } 1, \qquad i = 1, 2, ..., n \end{aligned} \tag{1.5}$$

The DCKP may be solved by an integer programming package such as CPLEX and LINDO, Yamada et al. (2002), to obtain optimal solution with limited size problem.

However, to deal with problems of up to 1000 items, Yamada et al. (2002) were the first to develop a B&B algorithm based on lower and upper bounds obtained respectively by an heuristic and a lagrangian relaxation of the conflict condition. Later, an algorithm combining lagrangian relaxation and pegging test for ordinary KPs was presented by the same author, Senisuka et al. (2005). Hifi and Michrafy (2006) proposed a metaheuristic approach combining a reactive local search algorithm and a tabu list, and developed in 2007 three different exact algorithms. Recently, Pferschy and Schauer (2009) treated the problem by mean of conflict graphs to derive pseudopolynomial algorithms. They assume that constraints are the edges and items are the vertices of a conflict graph. Fully polynomial time approximation schemes (FPTAS) were obtained from these algorithms.

❻ The Minimization Knapsack Problem (MinKP)

Problem aspect	Static and deterministic
Available information	Items features

All maximization problems have an equivalent minimization versions, which suppose to minimize the cost (instead of the profit in the maximization version) or to minimize the profit of the unloaded items. The exact algorithms of each maximization form yield also an exact solution for the minimization version, but approximate algorithms does not always work for the other version, Guntzer and Jungnickel (2000). Here we focus into the minimization version of the binary KP.

The MinKP is a transformed version of the KP01 to a minimization problem. The objective is to minimize the total profit of the unselected items while their combined weight is at least equal to $y = \sum_{i=1}^{n} w_i - c$.

$$\begin{aligned} &\text{Minimize} && Z(x) = \sum_{i=1}^{n} v_i y_i \\ &\text{Subject to} && \sum_{i=1}^{n} w_i y_i \geq y, \\ & && y_i = 0 \text{ or } 1; \qquad i = 1, 2, ..., n \end{aligned} \tag{1.6}$$

The binary variables y_i indicates the not loaded items in the maximization version. Therefore, to the maximization problem solution corresponds a solution of the MinKP, defined as $y_i = 1 - x_i$. Guntzer and Jungnickel (2000) have presented a number of ε-approximate greedy algorithms for the MinKP that corresponds also to the maximization version.

❼ The Stochastic Knapsack Problem (SKP)

Problem aspect	Static and stochastic
Available information	The number of items
Probability distribution	Known

Assuming that our knowledge about the input data is probabilistic; the set of items is given while the reward and/or the weight coefficients are independent and identically distributed random variables. The SKP knew several formulation, each depending on the author description.

Morton and Wood (1998) define the SKP as: a set of n items where associated weights are deterministic but returns are random with known distribution. The purpose is to maximize the probability that the overall return threshold is reached or exceeded.

This problem can be formulate as:

$$\begin{aligned} \text{Maximize} \quad & P(\sum_{i=1}^{n} u_i x_i \geq r) \\ \text{Subject to} \quad & \sum_{i=1}^{n} w_i x_i \leq c, \\ & x_i = 0 \text{ or } 1; \qquad i = 1, 2, ..., n \end{aligned} \tag{1.7}$$

where r is a deterministic return threshold. The authors presented a DP procedure to solve the SKP with independent normally distributed returns, and a Monte Carlo approximation procedure with general assumption on the random returns.
Kosuch and Lisser (2008) studied two variants of the SKP; the SKP with simple recourse and the SKP with a probabilistic constraint. They developed an upper bound using an B&B algorithm and solving a continuous subproblems. Besides, using a stochastic gradient method, they solved the continuous SKP with simple recourse, and a stochastic Arrow-Hurwicz algorithm to solve the constrained continuous SKP.

Table 1.2: Summary of Literature Review on Static KPs

			SOLUTION APPROACHES				APPLICATIONS
			EXACT		HEURISTIC		
KNAPSACK PROBLEMS	STATIC PROBLEMS	*KP01*	**DP**	▶ Bellman (1950), Fayad & Plateau (1975), Pisinger (1995), Martello & Toth (1997)	**PTAS**	▶ Capara et al. (1997)	Cargo Loading
			B&B	▶ Kolesar (1967), Martello & Toth (1980), Pisinger (1997)	**FPTAS**	▶ Lawler (1979), Magazine & Oguz (1981)	Capital budgeting
			DP and B&B	▶ Martello & Toth (1984), Plateau & Elkihel (1985)			Cutting stock
		BKP	**DP**	▶ Pferschy (1999)	**GREEDY**	▶ Martello & Toth (1990)	
			B&B	▶ Tamir (2009)	**PTAS**	▶ Ingargiola & Korsh (1977), Pisinger (2000), Kellerer et al. (2004)	
		SSP	**DP**	▶ Martello & Toth (1984), Pisinger (1999), Samo & Toth (2002)	**FPTAS**	▶ Kellerer et al.(1997)	Two processor scheduling
					LOCAL SEARCH	▶ Ghosh & Chakravarti (1999)	Cargo Loading
					GA	▶ Wang (2003)	
		MOKP	**DP**	▶ Klamroth & Wiecek (2000), Bazgan et al. (2009)	**MOEA**	▶ Zitzler & Thiele (1999) (2001), Alves & Almeida (2007)	
			B&B	▶ Visée et al. (1998)	**HYBRID MOEA**	▶ Knowles & Core (2000)	Assigning jobs to several
			LABELING ALG.	▶ Captivo et al. (2003)	**GA**	▶ Ben Abdelaziz et al. (2000)	machines
					TABU SEARCH	▶ Ben Abdelaziz et al. (2000)	
		DCKP	**B&B**	▶ Yamada et al. (2002)	**LAGRANGIAN LOCAL SEARCH and TABU LIST**	▶ Yamada et al. (2002)	
				▶ Hifi and Michrafy (2006)		▶ Hifi and Michrafy (2006)	
			CONFLICT GRAPHS	▶ Pferschy & Schauer (2009)	**FPTAS**	▶ Pferschy & Schauer (2009)	
		SKP	**DP**	▶ Morton & Wood (1998)	**MONTE CARLO**	▶ Morton & Wood (1998)	
			B&B	▶ Kosuch & Lisser (2008)			
			GRADIENT METH.	▶ Kosuch & Lisser (2008)			
		MinKP			**GREEDY**	▶ Güntzer & Jungnickel (1999)	

1.4.2 Dynamic knapsack problems

Problems considered so far are static and/or deterministic KPs, since items are considered at a point in time, and their weights and values are known beforehand. However in such dynamically changing environment, many practical applications arise with an evolutionary loading process; profits are not stable and supplementary costs are supported. The static and deterministic form of the KP cannot model suitably these problems. Therefore, it is more realistic to consider a dynamic process instead of the static one. This kind of problems refers generally to the dynamic KP (DKP).
A DKP occurs in such situations: offers arrive successively over time and their data are learned as soon as items arrive. The DM observes items one by one and have to make an immediate decision on the acceptance or the rejection of the current one.

As for the static version, researchers studied different variants of the problem. In this section, we will review some variant of the DKP. First we focus in the basic version of DKP. In a second subsection, we will investigate the DKP applied to auctions field. Then we will study dynamic and stochastic KPs. This section will be summarized by table 1.2.

❶ The Online Knapsack Problem (OKP)

Problem aspect	Dynamic and stochastic
Available information	The number of items
Probability distribution	Unknown

As its name indicates, in the OKP, items appear online, one by one. Item's weight and reward become known once the item is received. We have to decide each time whether to accept or to discard the current item, before observing the next one. If discarded, an item cannot be recalled any more. We are asked to load items which maximize the overall reward without exceeding a fixed capacity.

Internet advertising and keyword auctions for internet search engines are examples of real-world applications of the problem.

The OKP was studied for the first time by Marchetti-Spaccamela and Vercellis (1995). Authors proposed a greedy algorithm to solve the relaxed KP. The same algorithm was generalized to solve the OKP (Algorithm 1).

Algorithm 1: The Fixed Choice Online Algorithm $\mathbf{G}(\overline{\mathbf{d}})$

```
choose a value d̄;
while i ≤ n do
    if d_i ≥ d̄ then
        x_i ← min(1, c/w_i);
        c ← c − w_i x_i;
    else
        x_i ← 0;
    end
    i ← i + 1;
end
```

Algorithm's idea is simple: the decision to hold an item or to reject it depends mainly on its density value computed as follows:

$$d_i = \frac{v_i}{w_i}, \quad for \quad i = 1, ..., n \tag{1.8}$$

A threshold density $\overline{d}$ is fixed at the beginning, then when an item is received, its density is compared to the threshold value. If $d_i \geq \overline{d}$, the item is selected, and the knapsack capacity is updated. Note that the algorithm allows to load a fraction of an item if the entire item does not fit into the knapsack.

Lueker (1995) improved the algorithm of Marchetti-Spaccamela and Vercellis (1995), by reducing the difference between the optimum and the approximate solution value. Besides, several works were concerned with the removable OKP, in which researchers assume that a loaded item can be removed from the knapsack to place another one. We quote as examples: Iwama and Taketomi (2002) and Iwama and Zhang (2007) who presented constant-competitive algorithms. Han and Makino (2009) suppose furthermore that items are fractional and propose a greedy online algorithm.

❷ The Knapsack Auction Problem (KAP)

Problem aspect	Dynamic and stochastic
Available information	Number of bids
Probability distribution	Unknown

An auction is a process in which a buyer, given a limited budget, would like to purchase items from a given set of bids. Bidders desire to keep their items valuation private. Items arrive in an online fashion and differentiate by quality. The buyer's problem is to hold a subset of maximum quality.
Auctions theory has drawn attention because of the considerable number of applications that can be modeled as an auction problem. Besides, a number of recent research considered the application of the KP to auction design.
Using the KP terminology, the problem is stated as follows.

We consider a knapsack with a limited capacity c and n bidders. Each bidder has an object, wishing to place it in the knapsack. The valuation of the agent i of having his item in the knapsack is denoted v_i, while w_i denotes its required space in the knapsack. Sizes are the public values, whereas items' values are private. If bidder i wins, his object is placed in the knapsack and he has a profit v_i. Every bidder attempts to maximize his utility, computed as the difference between his valuation and his payment. The KAP is to maximize the knapsack contents value as much as

possible.

Aggarwal and Hartline (2006) studied the KAP for advertising in web page and broadcast bandwidth and give a constant factor approximation for the unlimited capacity knapsack. The proposed algorithm has the following structure:

Algorithm 2: Approximative KP Algorithm

1. Ignore large objects with $w_i > c/2$;
2. List the remaining objects in the order of decreasing value per unit size, $d_i \leftarrow v_i/w_i$;
3. Select the largest prefix of the object list that fits in the knapsack as the winner set;
4. Let d^* be the largest value per unit size of the losers;

Output $Z(x) \leftarrow d^*x$ for $x \leq c/2$ and ∞ otherwise;

Recently, Ensthaler and Giebe (2009) proposed an extension of the Dantzig's greedy heuristic for the KP01 to solve the KAP. Their algorithm operates as follows:

- Each bidder makes an initial bid.
- The buyer asks to bidders to discount their bids by a certain decrement; because of the limited budget of the buyer, he cannot purchase all bids.
- Bidders who refuse to lower their offer are discarded.
- The algorithm iterates until the budget covers the sum of active bids.
- Once the stopping criterion is satisfied, the buyer purchase active offers and bidders are rewarded about their final bids.

❸ The Knapsack Secretary Problem (KSP)

Problem aspect	Dynamic and stochastic
Available information	The number of candidates
Probability distribution	Unknown

The KSP is a weighted form of the secretary problem. Formally the problem can be stated as: n items (or secretaries), each with an associated weight w_i and reward v_i, are presented to the DM in a uniformly random order. The weight and reward distributions are not known a priori, and their values are learned at the arrival time. No recall is allowed; the decision should be immediate to accept the item or to discard it.

The problem was introduced by Babaioff et al. (2007), who evoked the similarities of the DKP with the secretary problem; if all items' weight are equal to 1, and the knapsack capacity is equal to k, the KSP is reduced to a multiple-choice secretary problem.

Authors proposed a $10e$-competitive algorithm for arbitrary weights and an e-competitive algorithm for the particular case where items have equal weights. For the weighted case, the algorithm was stated as follows.

Given the input set of items U, if $Q \subseteq U$ and $c > 0$, they determine the optimum fractional packing of elements of Q into the knapsack of capacity c. That is by solving the following problem:

$$
\begin{aligned}
&\text{Maximize} && Z(x) = \sum_{i=1}^{n} v_i x_i && \\
&\text{Subject to} && \sum_{i=1}^{n} w_i x_i \leq c, && \\
& && x_i = 0 && \forall i \notin Q \\
& && x_i \in \{0, 1\} && \forall i
\end{aligned}
\tag{1.9}
$$

The selection criteria is based on the density value (1.8) and a threshold density d_Q such that:

$$
x_i = \begin{cases} 1 & \text{if } \; d_i > d_Q, \quad \forall i \in Q \\ 0 & \text{if } \; \text{otherwise} \end{cases}
\tag{1.10}
$$

Many real-world applications involving the allocation of resources under uncertainty can be modeled as a KSP, such as: hiring employees and assigning job to machines.

❹ The Dynamic and Stochastic Knapsack Problem (DSKP)

Problem aspect	Dynamic and stochastic
Available information	The number of items
Probability distribution	Known

A DSKP arises when we are in charge of allocating a limited resource for a number of items. We receive items according to a Poisson process in time. Each item has an associated reward and a certain weight, which are random, and unknown before items arrival. However, the joint distribution of rewards and weights is known and

is independent of the arrival time and of other rewards. An item can either be accepted, and then loaded in the knapsack, or rejected. If the item is accepted, the associated reward is received otherwise, a penalty is incurred and the item cannot be reconsidered later. The DM may stop the process at any time, even before the deadline and the exhaustion of the resource capacity. Papastavrou and Kleywegt (2001) explain this by a practical example; in many practical situations, we have to dispatch vehicle even before attaining the deadline or the complete filling of the vehicle. The objective is to determine a policy for accepting items and for stopping the process which maximizes the overall reward.

The DSKP with deadlines was studied by Papastavrou et al. (1996). More recently, they studied special cases of the DSKP, assuming that all items have identical, Papastavrou and Kleywegt (1998), and random sized weights, Papastavrou and Kleywegt (2001). They showed that the optimal policy for the DSKP is a threshold type policy. Their decision rule was presented as follows:

Given a knapsack with capacity c and a sequence of items which arrive according to a stochastic process with T discrete periods ($t = \{1, 2, ..., T\}$). There is a probability p of arrival of an object in each period, and a probability $1 - p$ of no arrivals. The weight w_i and the reward v_i of an item become known once arrived. They are independent from other arrivals, positive, random variables, and are distributed according to a known joint distribution $F_{WV}(w, v)$. Papastavrou et al. (1996) proposed the following equation for the optimal expected accumulated value :

$$EV_t^c = \sup_{\pi \in \Pi} \{E[V_t^c|\pi]\} \tag{1.11}$$

Where $E[V_t^c|\pi]$ denotes the expected value accumulated, c is the capacity updated until the current period t, and Π the class of policies which recommend the loading or the rejection of the current object and which consider only the past events.

Equation (1.12) shows that the expected value accumulated from period t to T, does not depend on the complete history of the process, but only on t, c and π ($\pi \in \Pi$). The optimal decision rule derived was:

$$\pi^*(t,c,w,v) = \begin{cases} \text{accept if} \quad v + EV_{t+1}^{c-w} \geq EV_{t+1}^{c} \quad \text{and} \quad w \leq c, \\ \\ \text{reject if} \quad v + EV_{t+1}^{c-w} < EV_{t+1}^{c} \quad \text{or} \quad w \leq c. \end{cases} \tag{1.12}$$

The incoming item is accepted if the expected value accumulated is improved when accepting this item, if compared with the constant threshold EV_{t+1}^{c}.
The optimal accumulated value can be calculated recursively from:

$$\begin{aligned} EV_t^c =& P[W \leq c, V + EV_{t+1}^{c-W} \geq EV_{t+1}^{c}] \\ & \times E[V + EV_{t+1}^{c-W} | W \leq c, V + EV_{t+1}^{c-W} \geq EV_{t+1}^{c}] \\ & + \{P[V + EV_{t+1}^{c-W} < EV_{t+1}^{c}, W \leq c] + P[W > c]\} EV_{t+1}^{c} \end{aligned} \tag{1.13}$$

Hence, the optimal accumulated value is equivalent to the probability-weighted sum of the possible quantities presented in (1.12).
Several investment problems can be considered as a DSKP. It represents a suitable design of many real applications in freight transportation, scheduling of batch processors, selling of assets and in selection of investment projects, Papastavrou and Kleywegt (1998).

Table 1.3: Summary of Literature Review on Dynamic KPs

			SOLUTION APPROACHES		APPLICATIONS
KNAPSACK PROBLEMS	Dynamic Problems	***OKP***	**Linear Time Dynamic Alg.**	▶ Marchetti-Spaccamela & Vercellis (1995), Lueker (1995)	Internet advertising
			Deterministic Alg.	▶ Han and Makino (2006)	Keyword auctions
		KAP	**Constant Factor Approxim.**	▶ Aggarwal & Hartline (2006)	Broadcast bandwidth
			Greedy Alg.	▶ Ensthaler & Giebe (2009)	
		KSP	**Constant-competitive Alg.**	▶ Babaioff et al. (2007)	Hiring employees
			e-competitive Alg.	▶ Babaioff et al. (2007)	Assigning job to machines
		DSKP	**Threshold Type Policy**	▶ Jason Parastavrou et al. (1996, 1998, 2001)	Freight transportation Scheduling of batch processors Selling assets

1.5 General comments

The KP is one of the most renowned and extensively studied problems in the literature. We presented in the previous sections a brief review of KPs literature.
We distinguished two fundamental directions from the existing literature: static problems and dynamic problems. This separation was adopted because of versions differences in terms of description, applications and resolution methods.

Indeed, in the static version, items are considered in a point in time; all items are present simultaneously to the selection, and the DM possesses all needed information to decide on the loading. However, the dynamic family of problems has a lack of information about incoming items, which appears one by one to the evaluation. Generally, provided information about incoming offers is the total number. In such case, the DM has to make decision about the current item under uncertainty.

From a resolution point of view, several exact and heuristic algorithms has been designed over years to solve static KPs. DP and B&B are at base of the wide range of exact methods, while heuristics were developed from greedy method, tabu search, GA, etc. Conversely, there exist a little literature for dynamic problems. Besides no competitive online algorithms exist, Iwama and Taketomi (2002), except for some special cases if allowing the removability assumption. Besides, all static problems has a known formulation whereas dynamic problems do not have.

These turned out rather obvious, if we consider versions seniority. Indeed, we found static KPs papers in the literature since 1897, while the dynamic version was introduced the first time in 1995. The dynamic field of KPs remain a new discipline, which drawn a great deal of attention in the last years for its suitability to design sequential resource allocation problems.

Even application domains differ from the static to the dynamic problems. As we mentioned before, static applications belong to the transportation domain, capital budgeting, etc. While dynamic applications are in web advertising, auctions, broadcast bandwidth, which are the most active and evolutionary domains nowadays. Table 1.4 summarizes the evoked differences between static and dynamic KPs.

Table 1.4: Comparison Table of Static and Dynamic KPs

	KNAPSACK PROBLEMS	
	Static Problems	**Dynamic Problems**
Problems appearance	Since 1897	First time in 1995
Fundamental aspect	Offline problems; does not account for the element of time	Online problems
Information about arrivals	Full information	Lack of information
Items reception	Items are present simultaneously	Items are received one by one, over time
Formulation	A known formulation	No common formulation
Resolution procedures	Exact and heuristic methods: • DP • B&B • GA • Greedy method • etc.	• No constant competitive algorithm were found for the basic form (only for some special cases) • Approximative algorithms
Application domains	Transportation, capital budgeting, processor scheduling	Web advertising, auctions, broadcast bandwidth

1.6 Conclusion

In this chapter we reviewed the existing literature of the KP. We underlined the importance of the problem which showed to be suitable to model several real-world situations. This explains the significant number of variations which knew during years.

We noticed that the wide range of works was concerned by the static and determin-

istic variants. However, recently few researchers was interested by the stochastic and the dynamic variants with the intention to find a more convenient description of real situations and to respond suitably to the asked problem.
Furthermore, we found that exact algorithms for this problem are mainly based on DP and B&B. While the field of heuristics is vaster; greedy method, GA, PTAS, FPTAS, etc.

Chapter 2

Optimal Stopping Problems

2.1 Introduction

In practical situations, we are encountering situations requiring a rapid decision. As decision contains risk, the problem is when and how to come to a decision. Choosing time to take a given action is fundamental in several real-world problems such as: the problem of purchasing an asset, selection of projects and marriage problem. Those problems belong to the optimal stopping problems framework.

The optimal stopping problem (OSP) is a dynamic optimization problem having many applications in several applicative domains. Special examples that have been considered in the literature are the secretary problem, the problem of job search, assigning a job to a single machine and the problem of purchasing an asset.

An OSP arises when a DM is faced with the problem of finding the best offer among a known number of offers that arrives successively. The offers are evaluated one by one, and a unique offer is selected at the end of the process. At each stage of the process, the DM has to decide whether to accept the current offer or to discard it. The next offer is given once a decision of rejection is made about the current one.

If an item is rejected, it cannot be re-examined later. The DM requires a selected offer, so that if he arrives to the last offer, he will be obliged to accept it even if more suitable offers were observed so far.
The main objective of this problem is to maximize the probability of choosing the best alternative among those examined.

The OSP has been extensively studied in the literature for its suitability in modeling a wide range of sequential sampling problems. It has been extended and generalized in many other versions by the release of some basic assumptions. These variations concerned mainly the number of offers to select, the number of arrived item at the same time, the probability distribution of arrivals, the number of deciders, and others.
Morever, different objectives were considered in studying the problem: maximizing the probability of choosing the best offer (Lindley, 1961;Dynkin, 1963; Gilbert and Mosteller, 1966), minimizing the expected rank of the selected offer (Lindley, 1961; Chow et al., 1964).

In this chapter we will focus on the OSP. In the first section we will detail the problem, then we present the formulation in the following section. After that, we study some variants and extensions of the OSP in section 2.4. We close the chapter with an illustrative table covering current knowledge on this problem.

2.2 Notation

We adhere to the following notations throughout this chapter.

Symbols	***Explanations***
n	Number of offers
i	Stage number
k	Relative rank
k'	Absolute rank
r_i	Relative rank of the item i
a_i	Absolute rank of the item i
r_i^j	Relative rank of the i^{th} applicant on the j^{th} attribute
a_i^j	Absolute rank of the i^{th} applicant on the j^{th} attribute
π^j	DM's payoff on the attribute j
m_i	Number of alternatives in group i
M_i	The cumulative number of alternatives evaluated until the stage i
$U(k')$	Utility function based on the absolute rank k'
$V_s(i,k)$	Expected utility when stopping in the i^{th} stage with a relative rank k
$V_c(i)$	Expected utility of continuation from stage i
$V^*(i,k)$	Expected utility of the i^{th} offer having a relative rank k

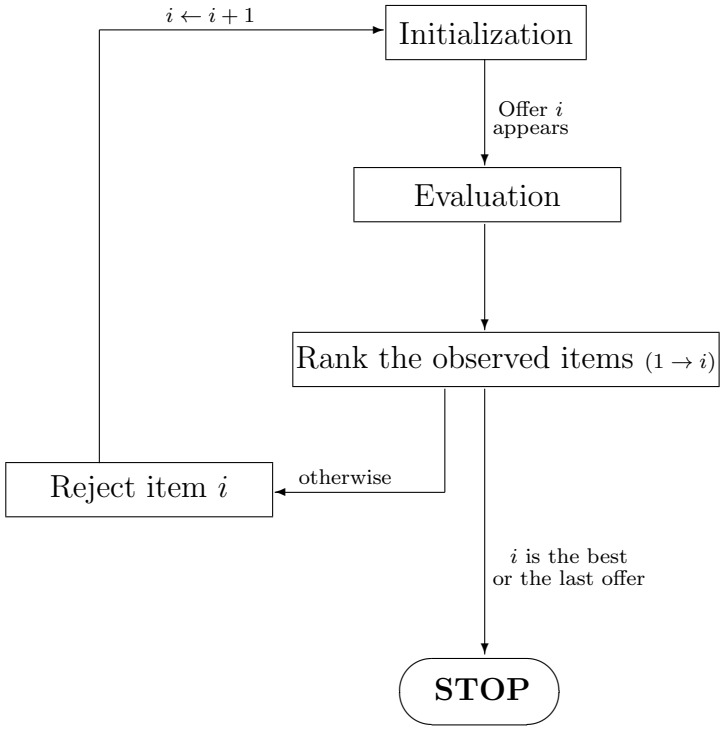
2.3 Statement of the problem

The OSP is a searching process, having as purpose to identify the optimal alternative among a sequence of offers arriving over time. The observer, without any prior information, evaluates the offers one by one. Once an offer is examined, the DM have to decide immediately whether to accept the offer, and stop the searching

process, or discard it, in favor of a possible better offer, and continue sampling.
He has to select at most one offer among all examined, and not allowed to quit the process without selecting an offer. If he arrives to the final offer, he has to accept it (even if the offer is not satisfactory).
At each stage of the process, the decision to be made is based on the value of the current offer and the possibility of finding a better alternative in a later stage.
The aim is to determine an optimal policy for stopping that maximizes the probability of choosing the best among all offers. The searching process is shown in figure 2.1.
The above description corresponds to the basic version of the OSP, known also as the secretary problem.
To summarize, an OSP is structured through the following assumptions, Ferguson (1989):

1. We are interested by a unique offer.
2. The number of offers is known.
3. Offers are evaluated one bye one, in a random order.
4. We can rank observed offers from the best to the worst without ties, and the decision is only based on these ranks.
5. The decision is irrevocable; a rejected offer cannot be recalled later.
6. We will be satisfied with nothing but the best offer.

Figure 2.1: Chart of the OSP



However, this basic form was extended in different directions giving arise to several variants of the problem. Each variant extend one or more of the classical assumptions, depending on the nature of the problem. Indeed, the common point in all variations formulation is that the decision is function of the ranks. We identify two types of rank:

- *The Relative rank*: is that attributed by the observer for each of the observed offers, and updated each time a new offer is received. Ranks are varying from 1 to the number of the current offer.

- *The Absolute rank*: is the position of an offer among all offers. This rank is available only on the last stage.

In what follows, we will review a number of extensions and generalizations of the OSP, present the related formulations and works.

2.4 Variants and extensions of the optimal stopping problem

The OSP constitutes a suitable model for dynamic selection problems. It covered a considerable number of applications, thanks to its flexible structure, allowing extensions and generalizations. Variants of the problem studied over years are relaxing or releasing one or more of the basic assumptions. Therefore, a number of additional assumptions were made:

- Instead of a known number of offers, a random and an infinite number of alternatives were considered.
- Different objectives: maximize the probability of selecting the best offer, minimizing the rank of the selected offer, select one of the best offers, etc.
- Different utilities has been adopted, depending on the description of the problem: nothing but the best utility, the zero sum utility, the minimum rank utility, etc.
- Two or more DMs
- The full-information case and the no-information case based on the availability of information on the distribution function of comers.
- etc.

In this section, we focus on some variants. First we review the basic version and present the relative dynamic formulation. Then we present a generalized form of

the problem with the objective to select one of the most good offers. The third variant studied the case of offers with a number of attributes. After that, we will be concerned by the problem modeling the situation of receiving a group of items per period. The last variant present the situation on which the decision involves more then a unique DM.

❶ The Secretary Problem (SP)

The SP is a special case of the OSP, where a number of candidates appears for a single secretarial position. Secretaries are interviewed one by one, and the interviewer is required to make an irrevocable decision; to accept the current candidate and stop accepting requests, or to refuse it in order to evaluate the next one. No additional specification is needed, hence we proceed to the formulation.

No value function is known. The only available information are relative ranks; the DM may rank alternatives from the best to the worst (the best offer has a relative rank=1, the next has rank=2, etc.). Hence, the decision to accept or discard an offer is based only on the relative ranks attributed. Since the DM's objective is to select the best among all offers, he wins 1 in term of utility if he selects the best offer, and 0 otherwise.

Thus we can formulate the utility function as:

$$U(k') = \begin{cases} 1 & \text{if } k' = 1 \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

Utility in (2.1) is a 0-1 payoff function, because the DM is satisfied if and only if, he selects the best alternative. If we vary the objective, other utilities implies.

Since the problem has a dynamic aspect, we need a dynamic formulation which includes the two parameters: the stage number i and the relative rank k of the i^{th} stage. The optimality equation of the DM is a recursive equation, stated as:

$$V^*(i,k) = \begin{cases} max\{V_s(i,k), V_c(i)\} & \text{if } k = 1 \\ V_c(i) & \text{otherwise} \end{cases} \tag{2.2}$$

Equation (2.2) indicates that the observer will be in the favor of the decision which maximizes his utility. That is the expected utility of an item is the maximum value between $V_s(i,k)$ and $V_c(i)$. Now the question is how and when to compute these values.

Stopping the search process is conditioned by the selection of the best offer, thus, the expected utility of stopping is computed in that only case. We denote by $V_s(i,k)$ the expected utility to stop the process by accepting the i^{th} offer, with a relative rank k. The utility of stopping is expressed as:

$$V_s(i,k) = \begin{cases} \dfrac{i}{n} & \text{if } k = 1 \\ \\ 0 & \text{otherwise} \end{cases} \tag{2.3}$$

On the other hand, if the DM chooses to continue sampling, we derive an expected utility of continuing.

$V_c(i)$ denotes the expected utility if we desire to continue the process from the i^{th} stage:

$$V_c(i) = \frac{1}{i+1} \sum_{k=1}^{i+1} V^*(i+1,k) \tag{2.4}$$

From the above DP formulation, we can derive the optimal strategy for the SP. Indeed, the optimal strategy is to discard the first $(i^* - 1)$ offers, and accept the first candidate thereafter satisfying:

$$\frac{1}{i^*} + \frac{1}{i^*+1} + ... + \frac{1}{n-1} < 1 < \frac{1}{i^*-1} + \frac{1}{i^*} + ... + \frac{1}{n-1} \tag{2.5}$$

The problem has been formulated alternatively using a markov chain approach (Dynkin, 1963).

❷ <u>One</u> out of the <u>r</u> <u>B</u>est (1RB)

Gusein-Zade (1966) proposed a generalized OSP, on which only assumption 6 is relaxed. Except for this one, the problem keeps the same assumptions (1-5) of the basic version. Indeed the DM is interested by one of the best offers, that is, an offer is accepted if its rank is at least equal to r.
Therefore, the utility is expressed as:

$$U(k') = \begin{cases} 1 & \text{if } k' \leq r \\ 0 & \text{otherwise} \end{cases} \tag{2.6}$$

The optimality equation is the same of that of the basic version:

$$V^*(i,k) = max\{V_s(i,k), V_c(i)\} \tag{2.7}$$

However, $V_s(i,k)$ measures in that case the probability that the i^{th} offer will have an absolute rank belonging to $\{k, k+1, ..., r\}$ given that its relative rank is about k. If the value of k overtake r, the DM loses.

Hence, we formulate the $V_s(i,k)$ as:

$$V_s(i,k) = \begin{cases} \sum_{r}^{j=k} \frac{C_{j-1}^{k-1} C_{n-j}^{i-k}}{C_n^i} & \text{if } k' \leq r \\ 0 & \text{otherwise} \end{cases} \tag{2.8}$$

The $V_c(i)$ keeps the equation of the basic OSP:

$$V_c(i) = \frac{1}{i+1} \sum_{k=1}^{i+1} V^*(i+1,k) \tag{2.9}$$

The optimal strategy is to select the current offer if $V_s(i,k) \geq V_c(i)$, and to continue to the next one otherwise.

The 1RB was studied identically by Gilbert and Mosteller (1966), Sakaguchi (1976) and Bojdecki (1978), for the full-information case, where incoming offers are from a known continuous distribution. Porosinski (1987) studied the 1RB for a random number of arrivals and the full-information continuous time 1RB, Porosinski (2003).

❸ The Multi-Attribute Secretary Problem (MASP)

The MASP is a generalization of the classical SP, where each offer is characterized by more than a unique attribute. Attributes refer to comparable dimensions of applicants. It may correspond to education, the work experience, degree of technical proficiency, etc. To simplify, an MASP has the following features, Bearden et al. (2005):

1. A known number of applicants for a single position. Applicants differ along k different and uncorrelated attributes.

2. Applicants are interviewed one by one, in a random order.

3. The DM assigns relative ranks for each of the applicant k attributes.

4. Once rejected, an applicant can never be recalled and we are not allowed to terminate the process without choose an applicant.

5. The DM earns a payoff of $\pi^j(a^j)$ per attribute j of the selected applicant (a^j denotes the absolute rank on attribute j; $1 \leq a^j \leq n$).

Since ranks are assigned for multiple attributes, a different definition of ranks is considered:

- The relative rank of the j^{th} attribute of the applicant i (r_i^j): is the number of applicant from 1 to i whose j^{th} attribute is at least as good as the i^{th}
- The absolute rank of the j^{th}'s attribute of the applicant i (a_i^j): is the number of applicant, among the n, whose j^{th} attribute is at least as good as the i^{th}'s

According to the above assumptions, the DM payoff of selecting the i^{th} applicant is written as:

$$\Pi_i = \sum_{j=1}^{k} \pi^j(a_i^j) \tag{2.10}$$

Given that $r_i = (r_i^1, ..., r_i^k)$ and $a_i = (a_i^1, ..., a_i^k)$, are respectively the set of relative and absolute ranks of item i, the expected payoff for selecting the i^{th} applicant is expressed as:

$$E(\Pi_i | r_i) = \sum_{j=1}^{k} \sum_{a_i^j = r_i^j}^{n} Pr(A_i^j = a_i^j | R_i^j = r_i^j) \pi^j(a_i^j) \tag{2.11}$$

Since at each stage, the decision is made only at base of the relative rank, the decision policy for each stage is a set of acceptable r_i of the current stage, denoted

R_i. Therefore, the global policy is the collection of stage policies $R = R_1, ..., R_n$.

The expected payoff of following the optimal policy is given by:

$$V_i^* = Q(R_i^*)E(\Pi_i|R_i^*) + [1 - Q(R_i^*)]V_{i+1}^* \tag{2.12}$$

This formulation leaded to the selection of a candidate of an intermediate quality, Bearden et al. (2005). However, other works are of interest. Gnedin (1981) studied the problem with the intention to select the applicant who is best on at least a unique dimension. In 1992, Ferguson extend the Gnedin's version, assuming that attributes can be dependent. He proved that the optimal policy has the same threshold form as basic SP, Ferguson (1992). Chotlos (1987) studied the MASP with the objective of minimizing the sum of two ranks for independent attributes.

❹ The Group Interview Problem (GIP)

In real-life, sequential sampling situations arise with a number of choices appearing over time. These alternatives is received in a random way. In the basic form SP, we assumed that offers are received one per period to simplify. However, it is more realistic to consider the possibility to encounter two or more alternatives at the same point in time.

The GIP is a generalization of the SP assuming that a sequence of groups of alternatives is received for evaluation over time. That is a group of candidates are inspected each time. The DM is required to select the best among all offers, the decision is irrevocable and he is not allowed to quit the process without a selection. Chun studied different form of the GIP. We will be concerned by the basic one, and we adopt Chun's formulation, Chun (2001).

Let G denote a group interview problem, in which a sequence of n groups is presented for the evaluation successively. Note that the i^{th} group contains m_i offers

($G = \{m_1, m_2, ..., m_n\}$). At the i^{th} stage, the i^{th} is evaluated. The relative rank corresponds in this case, to the position of the offer among all evaluated so far.
To maximize the expected utility, the following recursive equation implies:

$$V^*(i,k) = \begin{cases} max\{V_s(i,k), V_c(i)\} & \text{if } i = 1, 2, ..., n-1 \\ U(i) & \text{if } i = n \end{cases} \tag{2.13}$$

$U(i)$ denote the utility of choosing the choice with an absolute rank i. The expected utilities when stopping and when continuing are given by:

$$V_s(i,k) = \sum_{a=k}^{M_n - M_i + r} U(a) \frac{C_{k-1}^{a-1} C_{M_i-k}^{M_n-a}}{C_{M_i}^{M_n}} \quad k = 1, ..., M_{i-1} + 1 \tag{2.14}$$

$$V_c(i) = \sum_{k=1}^{M_i+1} V^*(i+1,k) \frac{C_{m_{i+1}-1}^{M_{i+1}-k}}{C_{m_i+1}^{M_i+1}} \quad i = 1, ..., n-1 \tag{2.15}$$

M_i denote the cumulative number of alternatives evaluated until the stage i, including those arrived at the i^{th} stage. Chun (2001) presented a boundary stage method as the optimal selection strategy for the GIP. He demonstrate further, that his previous works on SP, the best choice problem and the minimum rank problem, are special cases of the discussed version. Chun (1999), studied the GIP in the full-information case and proposed a dynamic programming technique to maximize the probability of selecting the best choice. The proposed decision rule is to select the best offer in the current group if its value is greater than the pre-specified decision value for that group.

❺ The Weighted Secretary Problem (WSP)

Variants of the OSP discussed until now, have a common property: no information is available about incoming offers. Therefore, the decision was based only on the relative ranks. However, in real situations, an interviewer may collect some information about each applicant. Different versions were studied in the literature. Here we focus on the WSP, introduced by Chun (1996). Indeed, the author suppose that given a prior information about applicants, the interviewer attributes a weight for each applicant, according to its credentials. From this background the WSP was structured.

Formally, a DM observers a sequence of offers, each with an associated weight in order to select the best one. The objective is to determine the optimal selection strategy, which maximizes the probability of choosing the best offer.

Let $W = \{w_1, w_2, ..., w_n\}$, $w_i \geq 0$, be the set of weighted associated to applicant before the evaluation. We distingue also the cumulative weight until the current stage i, expressed as $W_i = w_1 + w_2 + ... + w_i$. According to the above assumptions, a dynamic formulation was proposed, Chun (1996).

Since our interest is in maximizing the probability of selecting the best offer, the maximum winning probability earned if selecting offer i is given by:

$$\pi^*(i) = max\{\pi_s(i), \pi_c(i)\} \tag{2.16}$$

The wining probability if we decide to select the i^{th} offer is expressed by:

$$\pi_s(i) = \frac{W_j}{W_n}, \quad \text{for } i = 1, 2, ..., n \tag{2.17}$$

The wining probability if we continue to the next candidate is:

$$\pi_c(i) = \sum_{k=i+1}^{n} p_{ik}\pi^*(k), \quad \text{for } i = 1, 2, ..., n-1 \tag{2.18}$$

The transition probability from stage i to stage k, p_{ik} is the following equation:

$$p_{ik} = \frac{W_i}{W_{k-1}} \frac{w_k}{W_k}, \quad \text{for } 1 \leq i < k \leq n \tag{2.19}$$

According the discussed formulation, the optimal selection strategy is to select the current candidate if $\pi^*(i) = \pi_s(i)$.

The full information case was well studied, but in other directions. Different problems were studied assuming that offers follows a known probability distribution: Uniform (Albright, 1977; Bruss and Ferguson, 1993); Normal (Albright, 1977; DeGroot, 1968), Poisson (Cowan and Zabczyk, 1978; Gnedin, 1996).

❻ The OSP with Multiple Decision Makers

The OSP described so far involves a single DM searching through a sequence of alternatives until a choice is made. In some real situations, the decision to take involves more than a single DM. However, making a common decision isn't a simple task, within a group of DMs. Each member of the group, has his own evaluation of a given offer, which will not correspond necessarily to the other members. This situation gives arise to many conflicts between the different DMs. Therefore, this version is considered as game between members, in which we ask for an equilibrium. In the rest we will interest for the OSP with two DMs. The bilateral optimal stopping problem (BOSP) occurs with two DMs observing a sequence of n offers, sequentially, one at a time. Different objectives were considered, including:

- The two DMs are about selecting the same offer
- Each DM is required to select his offer independently

- Two players possessing different degrees of information; the first player knows the value of n before seeing the first observation, while the other player knows only the distribution of n, but not its actual value, Abdelaziz and Krichen (2005).

Sakaguchi (1984) studied the BOSP with a no-choice option, and proposed the following utility for player I:

$$\begin{cases} 1 & \text{if I accepts the best offer or II does not accept the best offer} \\ -1 & \text{if I does not accept the best offer or II accepts the best offer} \\ 0 & \text{if no selection is made for both players} \end{cases} \tag{2.20}$$

According to the utility in (2.20), the objective of player I is to maximize his utility, whereas player II aims to minimize the expected payoff of player I. Moreover, this problem is known as a *zero sum game*; the accumulated utilities of the two players is equal to 0. The priority is assigned to player I, that is player I decides the first on the current offer, and if he chooses to discard it, player II has the possibility to accept it, or to discard it definitively.
The recursive equation of player I at stage i is given by:

$$V^*(i,k) = max\{V_s^1(i), V_c^1(i)\} \tag{2.21}$$

The expected utility of accepting the best offer is stated as:

$$V_s^1(i)) = 2\frac{i}{n} - 1 \tag{2.22}$$

$V_c^1(i)$ measures the expected utility of selecting the best offer in a later stage, or the expected utility when II accepts an offer which is not the best.

The relative equation is:

$$V_c^1(i) = min\{1 - 2\frac{i}{n}, \sum_{j=i+1}^{n} \frac{i}{j(j-1)} V^*(i,k)\} \tag{2.23}$$

The optimal strategy for player I, derived by Sakaguchi (1984), is: to discard the first $b^* - 1$ alternatives. Player II accepts the first alternative with a relative rank 1 thereafter, and which is before the $(a^* - 1)$st offer. Player I accepts the first offer having a relative rank of 1 after a^*, where a^* is the smallest integer $\geq n/2$ and b^* satisfying:

$$\sum_{j=b-1}^{n-1} j^{-1} \geq 2\sum_{j=a^*}^{n-1} j^{-}1 - \frac{n-1}{a^*-1} + \frac{3}{2} > \sum_{j=b}^{n-1} j^{-1}$$

The OSP with two DMs was studied by Abdelaziz and Krichen (2005) in no-information case, and with the objective to derive a compromise choice for the two DMs. Immorlica et al. (2006) studied also the OSP with two DMs and extend their study to the multiple DMs case. They proposed an equilibria of the induced game based on the Nash equilibrium.
Several authors interested in the problem when each DM is required to select his own offer, as: Fushimi (1981) and Sakaguchi (1984, 1985).

❼ Other Variants

In what precedes, we detailed some variants of the well known OSP. However, OSP literature is larger. Table 2.1 regroups several other generalizations and extensions of the problem.

Table 2.1: Summary of some OSP variants

OSP variants	Description	Researchers
Multiple-choice SP	A SP in which the DM is allowed to select more than a unique offer	Preater (1994) Assaf et al. (2000, 2002, 2004) Kleinberg (2005)
Matroid SP	The elements of a matroid appears in an online fashion. The DM must decide whether to accept the current element or to discard it taking into account that the set of chosen elements should be an independent set in the underlying matroid. The objective is to maximize the combined value of the selected elements	Babaioff et al. (2007)
The Discounted SP	A SP with a discount time-dependent factor $d(t)$. The profit of holding an offer i at time t is $d(t) \cdot v(i)$	Rasmussen and Pliska (1976) Babaioff et al. (2009)
SP with Uncertain Employment	An OSP with probabilistic availability of offers; any item, if accepted, has some probability of not being available, in which case it has to be passed over and the next item	Smith (1975) Petrucelli (1981) Lehtinen (1993) Ano et al. (1996)
OSP with recall	Allow the observer at any stage to go back and try to accept an item which had been previously rejected. If it is available it is accepted, otherwise the observer must continue inspecting new items.	Yang (1974) Petrucelli (1981)
OSP with unknown number of objects	An OSP with a random number of objects	Stewart (1981) Abdel-Hamid et al. (1982) Horiguchi and Yasuda (2009)

2.5 Conclusion

We outlined in this chapter the OSP, the common name of the sequential sampling problems. It constitutes a field of study within mathematic-probability-optimization. Literature in OSP is extensive, such as its variants.

We presented in this search some of the most known variations. Extensions of the problem goes in different directions, adding new assumptions to the basic version or removing some ones.

The Optimal Stopping Knapsack Problem

Chapter 3

The Optimal Stopping Knapsack Problem

3.1 Introduction

In dynamic environments, decision-making is a vital action to guarantee competitiveness. Often decision-making is designed as a searching process aimed to identify the better action in the best time. Since the number of possible alternatives is limited and the time constitutes also a constraint, the DM has to act as quickly as possible.

We can cite some practical applications that require a sequential selection of alternatives as: the problem of adopting technological innovation, hiring an employee, selecting projects, selling of assets and investment of funds.

These kinds of problems belong to the resource allocation problems family, where the main objective is to dynamically assign a limited capacity to offers arriving over time, with random sizes and random values. Here, we may appeal to the DKP.

A DKP holds when we are searching among a finite set of offers, in order to select a satisfactory subset. Offers arrive over time without any prior information, and are

characterized by a specific reward as well as a specific weight. The main purpose is to model a process evolving in time, searching among a sequence of offers in order to carry the best possible subset which: maximizes the overall value of the selected offers and exhausts the capacity constraint.

In this search, we propose a new variant of the DKP, namely the optimal stopping knapsack problem (OSKP), which allows the delay of items for next stages; discarded items are not lost, and can be reconsidered in next stages. However, delayed item are penalized by a decrease of utility.
The main dilemma in this problem consists in deciding whether to load a currently observed item or delay it to next stages. We propose a dynamic strategy for the DKP managed by an optimal stopping rule. The proposed approach controls the loading process stage by stage, evaluates each incoming item at the arrival time and attributes a new ranking of already observed items. Based on these ranks, we derive the item's utility value, required for the dynamic calculation. The stage's dynamic computation determines which items to load in the knapsack and those should be delayed. At each stage of the dynamic process, items that can be packed are inserted into a static Knapsack sub-problem solved using the simplex method. Our approach is run on a large test bed of DKPs with sizes belonging in {10,1000}. We also develop some metrics to show the effectiveness of our results.

This chapter is split of four sections: the first one constitute a description for the studied problem. In a second section we listed the notations used to formulate the problem in the third section. Finally we detail the proposed solution approach.

3.2 Statement of the problem

We consider a set of items, from which we have to select a subset to be packed in a knapsack. The knapsack has a limited capacity. Items arrive randomly, over n discrete periods, without any prior data, and it joins the evaluation process once arrived. Each evaluation has to lead to an immediate decision; to accept the item or to leave it. Once an item is selected, the associated reward is received and it can not be taken out of the knapsack. The leaved items can be selected in a later stage; if an item is not chosen at the arrival time, it joins the set of item that will be evaluated in the next stage.
The utility of the item is discounted each time the item is rejected, until it will be packed. Penalties is considered in this problem to enforce the DM to take a closely decision.
We are looking for an optimal policy for accepting items and for stopping the search process that maximizes the overall reward, under the capacity constraint. The decision process involves a single DM.

Therefore, a DKP is considered and the decision to make at each stage is based on the dynamic computation of the expected utilities, when accepting the item and when delaying it. Figure 3.1 resumes the selection process.

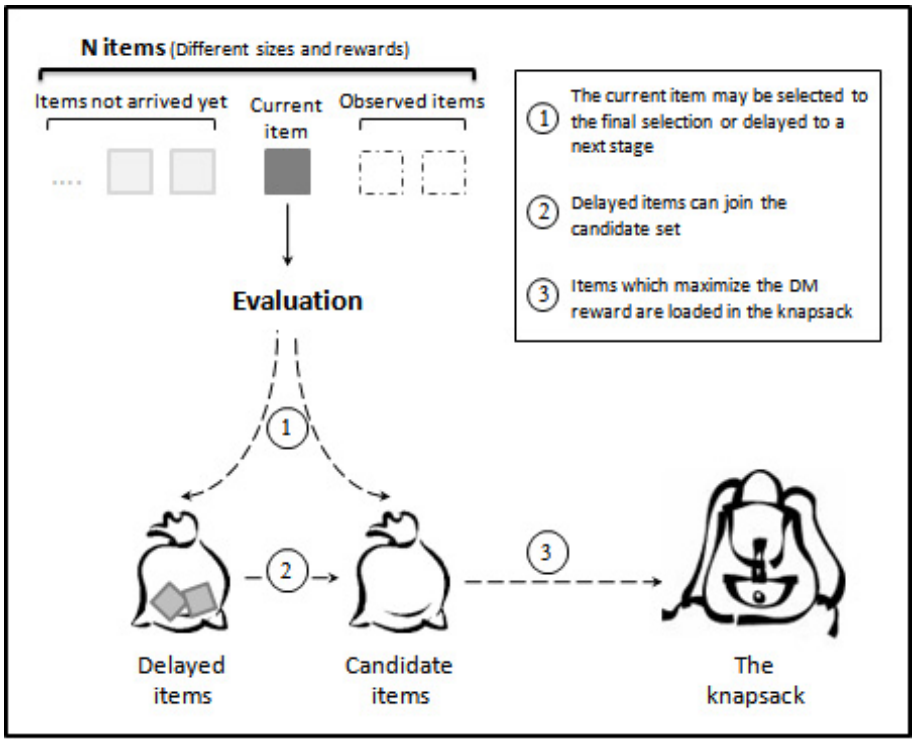


Figure 3.1: Decision Process of the OSKP

3.3 Notation

A summary of notation is given in the next table.

Symbols	***Explanations***
n	Number of items
m	Number of loaded items
i	Item number
j	Stage number
k	The relative rank

k_a	The absolute rank
C	The knapsack capacity
c_j	The remaining knapsack capacity at the j^{th} stage
w_i	Weight of item i
v_i	Value of item i
R_i	The ranking ratio
S	The set of candidate items
$U^i(k_a)$	The utility function based on the absolute rank k_a
$EU^i_s(j,k)$	The expected value when stopping of the item i at the j^{th} stage when its relative rank=k
$EU^i_c(j)$	The expected value when continuing of the item i at the j^{th} stage
$EU^i(j,k)$	The expected value of the item i at the j^{th} stage when its relative rank=k
$f(k_a\|k,j)$	The probability of having k_a given k, at the j^{th} stage
$x_i \in \{0,1\}$, $i = \{1,...,n\}$	If item i is selected $x_i = 1$, otherwise $x_i = 0$

3.4 Mathematical formulation

We are given n items arriving online, one at a time, for a knapsack of limited capacity C. Each item has a specific weight w_i as well as an associated value v_i. No information about arriving items is available. The only provided information is the total number of items n.

Note that the above description has few similarities with the OSP; an agent is receiving a number of offers arriving over time in a random order, without a prior information, in order to select the best one.

Taking into account this resemblance, we may appeal to the OSP terminology to formulate our problem. Therefore, the problem can be viewed as a decision process aiming to identify the subset of the best offers among the stream of offers arriving successively. Whenever a new offer appears for the competition, the DM updates the awarded ranks at the previous stage with regard to the current offer. This ranking

approach is aiming to states the new item among the previous one.
Our model is based on four components:

- $U^i(k_a, j)$: expresses our interest in i if it has an absolute rank=k_a at the j^{th} stage
- EU^i_s: is a prediction value of the i's utility, if we choose to stop at this item; to nominate i for the final selection
- EU^i_c: is the expected utility of i if we choose to continue to the next stage; to delay the decision about item i for next stages
- EU^i: is the best expected utility value among the EU^i_s and EU^i_c; decides on the load or the delay of i
- $f(k_a|k, j)$: represents the probability that i is of k_a given k, at j

In the remainder of this section, we detail our model formulation.

Unlike to the OSP, where the DM is only satisfied by the best among all offers, we are looking for the best subset to be packed in the knapsack. Thus, we have to adopt a different utility function.
Indeed, we will appeal to two different utility functions: $U_1(k_a)$ and $U_2(k_a)$.

❶ The inverse-rank utility

Proposed by Chun (2001), and expressed as:

$$U^i_1(k_a) = \frac{1}{k_a} \tag{3.1}$$

Hence, if the DM selects the best offer he wins 1, $\frac{1}{2}$ if he select the second best offer, and so on. This utility function is valid only if the selected item is just arrived.

However, if the item has been rejected in an earlier stage, a penalty in term of utility is supported. We assume in this case that the utility is discounted to the utility of the next rank, each time the object is rejected. That is:

$$U_1^i(j, k_a) = \frac{1}{k_a} \times \prod_{p=1}^{j-i}(1 - \frac{1}{k_a + p}) \tag{3.2}$$

According to equation (3.2), the utility of an item arriving at the second stage and delayed to the 4^{th} stage, is computed as: $U_1^2(4, 1) = 1 \times (1 - \frac{1}{2}) \times (1 - \frac{1}{3}) = \frac{1}{3}$.

❷ The regressive fraction utility

We propose a second utility function, given by:

$$U_2^i(k_a) = \frac{n - k_a + 1}{n} \tag{3.3}$$

The utility function $U_2^i(k_a)$ attributes a fraction of n to item i given k_a, e.g. if we have $n = 5$, the best item utility is 1, the second $\frac{4}{5}$, the third $\frac{3}{5}$, etc.

As for the first utility function, a penalization is incurred in case of delay. The utility function becomes:

$$U_2^i(j, k_a) = \frac{n - k_a + 1}{n} \times \frac{n - (j - i)}{n} \tag{3.4}$$

The first ranked item, which has been delayed twice from stage 2 to stage 4, has the utility: $U_2^2(4, 1) = 1 \times \frac{5-(4-2)}{5} = \frac{3}{5}$.

Table 3.1 shows a comparison between the utility functions outcomes for $n = 5$.

The experimental study goes to demonstrate later the results behavior for each of the two equations. Now that we defined the utility functions, we proceed to the expected utilities formulation.

Table 3.1: Comparison of utility functions values

Utility Functions | k_a		1	2	3	4	5
$U_1^i(j,k_a)$	New comers	1	$\frac{1}{2}$	$\frac{1}{3}$	$\frac{1}{4}$	$\frac{1}{5}$
	Delayed items	$\frac{1}{2}$	$\frac{1}{3}$	$\frac{1}{4}$	$\frac{1}{5}$	$\frac{1}{6}$
$U_2^i(j,k_a)$	New comers	1	$\frac{4}{5}$	$\frac{3}{5}$	$\frac{2}{5}$	$\frac{1}{5}$
	Delayed items	$\frac{4}{5}$	$\frac{16}{25}$	$\frac{12}{25}$	$\frac{8}{25}$	$\frac{1}{6}$

The expected utility of accepting an item (EU_s) is calculated by taking the probability-weighted sum of all possible outcomes under certain circumstances, that this item may have any of the possible ranks.

$$EU_s^i(j,k) = \sum_{k_a=k}^{n-j+k} U^i(j,k_a) f(k_a|k,j), \quad \text{where} \quad f(k_a|k,j) = \frac{\binom{k_a-1}{k-1}\binom{n-k_a}{j-k}}{\binom{n}{j}} \tag{3.5}$$

In equation (3.5), $U^i(j,k_a)$ is replaced by either function of utility, discussed so far. While the expected utility of rejecting an offer (EU_c) is the following:

$$EU_c^i(j) = \begin{cases} \dfrac{1}{j+1}\displaystyle\sum_{k=1}^{j+1} EU^{i*}(j+1,k) & if \;\; j < n \\ \\ 0 & otherwise \end{cases} \tag{3.6}$$

Equation (3.6) indicates that arriving to the last stage, no item should be delayed more. All remaining items are nominated for the final selection.
Since we are seeking the optimal subset of items, we are satisfied by nothing but items which maximize our utility.

Therefore, the expected utility of the i^{th} item at the next stage $(j+1)$ is given by:

$$EU^{i*}(j+1,k) = \max[EU_s^i(j+1,k), EU_c^i(j+1)] \tag{3.7}$$

The decision rule is based on the following inequality:

$$EU_s^i(j,k) \geq EU_c^i(j) \tag{3.8}$$

If the inequality (3.8) holds, the item i is a candidate, and joins the candidate set S, otherwise i is delayed to the next stage. Once ranked is attributed and the expected utilities are calculate for all present item, the candidate set is formed.

At each stage, a static KP having as input the set of candidates and constrained by the remaining capacity of the knapsack until the current stage c_j, is solved.
Following is the subproblem KP_j:

$$\begin{aligned} &\text{Maximize} && Z(x) = \sum_{i \in S} v_i x_i \\ &\text{Subject to} && \sum_{i \in S} w_i x_i \leq c_j \end{aligned} \tag{3.9}$$

3.5 The proposed solution approach

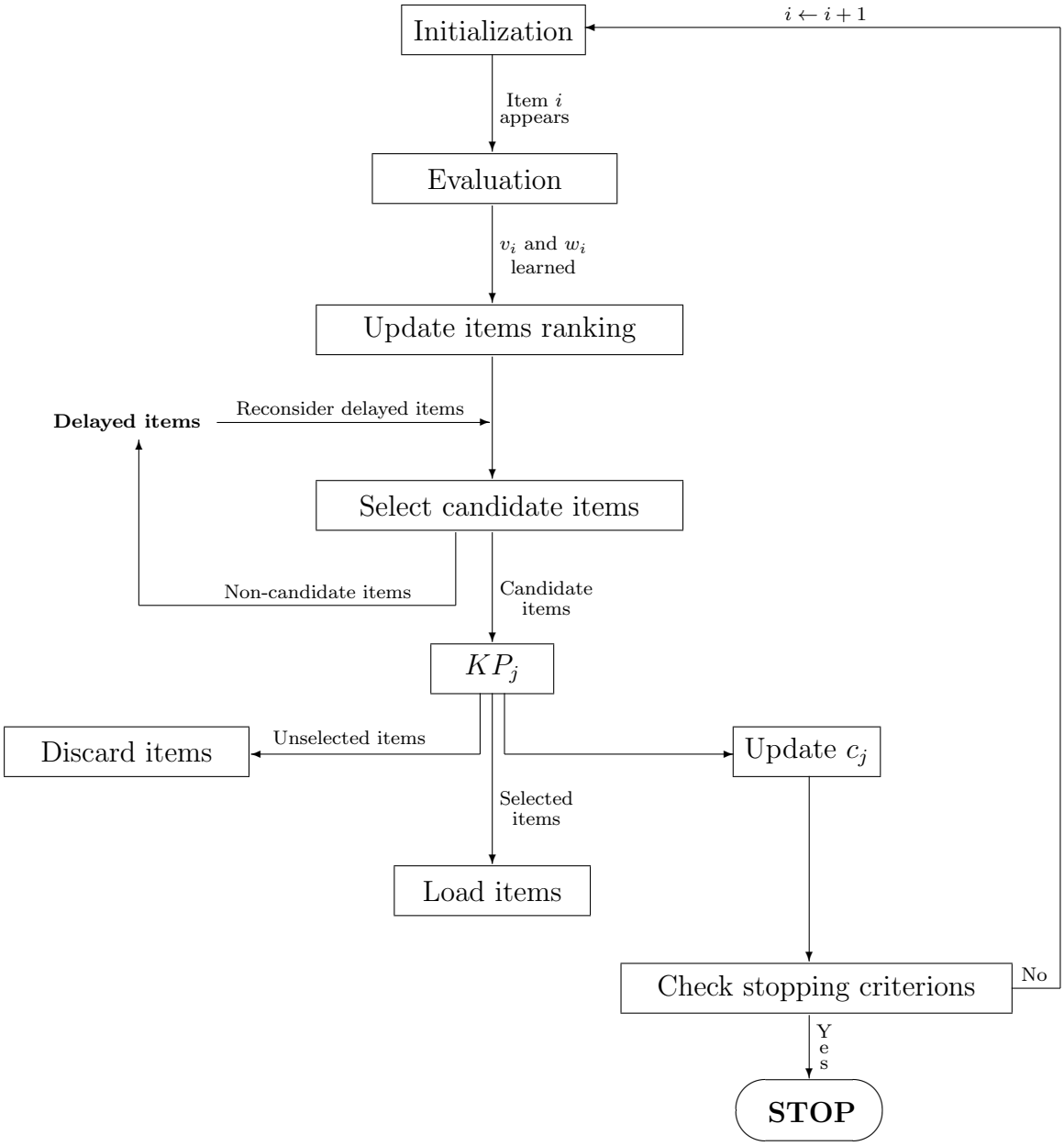
The OSKP asks for the best subset of items arriving randomly over time. As we are seeking for the optimal solution, we developed an exact algorithm to solve the problem. Stage by stage, and based on a ranking function, we decide on the loading and the delay of the incoming items.

However, it is relevant to see the algorithm behavior for the same stream of items, but for different ranks, i.e. if we present the same set of items to two DMs, often they will not have the same interest in items and each of them will attribute an order different from the other one. Therefore, we present a second algorithm, which generates a decision strategy for each problem, that still valid for different items valuations. In the rest of this section, we detail how our algorithms operate. We present both algorithms accompanied by examples.

❶ The Optimal Solution Algorithm (OSP)

Initially, the algorithm receive as inputs the total number of items and the knapsack capacity. Items are presented one per stage. Upon the arrival of a new item, the algorithm proceed to the ranking in order to arrange items by density.
The expected utilities of available items are computed thereafter, based on the attributed ranks. By means of the stopping rule, candidate items are identified. Once all available items are evaluated and the candidate set is formed, the algorithm appeal to the KP_j method to solve a KP01, given the candidate set and the remaining capacity until the current stage. Resulting items are loaded in the knapsack and the remaining are rejected definitively.
The knapsack capacity is updated and the algorithm reiterate until the capacity is exhausted or all items are observed.
Figure 3.2 resumes the approach progress, step by the step. The algorithm will be drawn thereafter.

Figure 3.2: Chart of the OSKP



The OSA

Algorithm 3: The Optimal Solution Algorithm of the OSKP

Input: The number of items n

Output: The optimal subset of items

Generate randomly and uniformly n coefficients;

$j \leftarrow 1$;

while $j \leq n$ **do**

  Rank the observed items from 1 to j;

  while $i \leq j$ **do**

    Compute the expected utilities of item i;

    if $E_s^i \geq E_c^i$ **then**

      $S \cup i$ /*S is the set of candidates*/;

    end

    $i \leftarrow i + 1$;

  end

  Solve $KP_j(S, c_j)$;

  Load the selected items;

  Update c_j

  if $c_j = 0$ **then**

    Quit the procedure;

  else

    $j \leftarrow j + 1$;

  end

end

An illustrative example

In order to explain the established algorithm, we demonstrate through the following example, how the proposed approach is applied during the selection process.
We consider, in the following table, a stream of 5 items arriving in time.

Table 3.2: Example with $n = 5$

		Weights	Values
5 items	O_1	9	100
	O_2	10	150
	O_3	7	120
	O_4	13	200
	O_5	15	250
$\hookrightarrow$ for a limited resource (C=40)			

At the beginning, our knowledge is limited to the total number of incoming items n and the knapsack capacity C. Items features are shown to the algorithm as soon as the previous stage is achieved. The algorithm commence by computing the expected utilities of the n items, each for its arrived stage and for all possible ranks. Note that the computation of the expected utilities in this example will be provided by the utility function U_2^i (3.4). Following are the expected utilities values:

Table 3.3: Expected utilities for $n = 5$

		Stages				
		5	**4**	**3**	**2**	**1**
Relative ranks	**1**	(1.0,0.0)	(0.96,0.6)	(0.9,0.72)	(0.8,0.78)	(0.6,0.79)
	2	(0.8,0.0)	(0.72,0.6)	(0.6,0.72)	(0.4,0.78)	_
	3	(0.6,0.0)	(0.48,0.6)	(0.3,0.72)	_	_
	4	(0.4,0.0)	(0.24,0.6)	_	_	_
	5	(0.2,0.0)	_	_	_	_

Table 3.3 presents the $EU_s^i(j,k)$ and $EU_c^i(j)$, for all $i \in \{1,5\}$, $j = i$ and $k \leq j$. Data in table's cells are presented according to the notation: $(EU_s^i(j,k_a), EU_c^i(j))$. Based on these information, we decide on the load or the delay of each item at its arriving stage. We can read from the table: item O_1 appears to the process at the first stage, its EU_c^1 is greater than the EU_s^1, that is, the algorithm prefer to continue to the next stage without packing it.

At the second stage, item O_2 can be packed if and only if its relative rank = 1, and so on. The above table is sufficient to learn about the current item, but not for the delayed ones. Therefore, beside to the initial table, we calculate the expected utilities for an delayed item.

Being in the second stage, we have delayed the item O_1 from the first stage. Its expected utilities values are written as: $(EU_s^1(2,k_a), EU_c^1(2)))$.

We appealed here to the ranking ratio:

$$R_i = \frac{v_i}{w_i} \tag{3.10}$$

R_i supplies the following order:

	O_1	O_2
Rank	2	1

Given the relative ranks, the algorithm calculates the expected utilities through equations (3.5) and (3.6), and return $(EU_s^1(2,2), EU_c^1(2)) = (0.32, 0.78)$.

According to the decision rule (inequality (3.8)), values are compared and the item O_1 is delayed for the second time. Equivalently, item O_2 is considered as a candidate for the KP_2 subproblem.

KP_2 allows the loading of O_2 in the knapsack. At the following stage, the algorithm receive the new item, O_3, and complete the calculation with a capacity discounted by the weight of the loaded item ($c_3 = 30$). Evenly, the rest of stages are treated and hence, items O_3, O_4, O_1 are loaded respectively in the third, forth and fifth stage.

At the end of the process, the knapsack contains the following items: $\{O_2, O_3, O_4, O_1\}$. Following are the obtained results:

– Accumulated reward: 570

– Remaining capacity: 1

– Number of loaded items: 4

❷ The Decision Strategy Algorithm (DSA)

This second algorithm attempt to derive all possible collections of items taking into account all possible ordering of items. Following are the algorithm and the associated example.

Algorithm 4: The Decision Strategy Algorithm of the OSKP

Input: The number of items n

Output: A decision strategy

Generate randomly and uniformly n coefficients;

$j \leftarrow 1$;

while $j \leq n$ **do**

  for $i \leftarrow 1$ **to** j **do**

    Compute the expected utilities of item i for all possible ranks;

    Identify candidates and possible sets;

    Solve a KP for each set of candidates;

    Load the possible collections of items;

  end

  $j \leftarrow j + 1$

end

An illustrative example

We deal with the same example in table 3.2. First, we draw the optimal stopping table with 5 items.

Table 3.4: Expected utilities for $n = 5$

		Stages				
		5	**4**	**3**	**2**	**1**
Relative ranks	**1**	(1.0,0.0)	(0.9,0.45)	(0.78,0.56)	(0.64,0.63)	(0.45,0.64)
	2	(0.5,0.0)	(0.43,0.45)	(0.35,0.56)	(0.27,0.63)	-
	3	(0.33,0.0)	(0.28,0.45)	(0.22,0.56)	-	-
	4	(0.25,0.0)	(0.21,0.45)	-	-	-
	5	(0.2,0.0)	-	-	-	-

Stage 1:

Item O_1 appears. We calculate the EU when stopping and when continuing. From the table 3.4, the $EU_c^1 > EU_s^1$ so the decision will be to continue to the next step without loading the item.

Stage 2:

O_2 arrives and O_1 is waiting from the previous stage.

	$EU_s^i(2, k_a)$		EU_c
	$k_a = 1$	$k_a = 2$	
O_1	0.37	0.21	0.63
O_2	0.64	0.27	0.63

Two cases can hold: if the object 2 is ranked the first, it is a candidate, otherwise it is not.

If O_2 is a candidate: $w_2 < c_2$ we can load item 2 $\Rightarrow K = \{O_2\}$, $c_2 = 40 - 10 = 30$. Otherwise no item is loaded.

Stage 3:

O_3 arrives. O_1 and O_2 are also present to the evaluation in this stage. Calculating the expected utilities, we found O_3 the unique candidate for the loading, if it is ranked first.

Stage 4:

O_3 and O_4 are candidates if and only if they obtain the rank 1.

Stage 5:

In the last stage, all items are already received. We do not need to calculate the expected utilities; all items are candidate for the selection. Therefore, we solve the KP for possible cases.

The tree in figure 3.3 resumes the decision strategy, stage by stage. At each stage, we represent the possible loads of items.

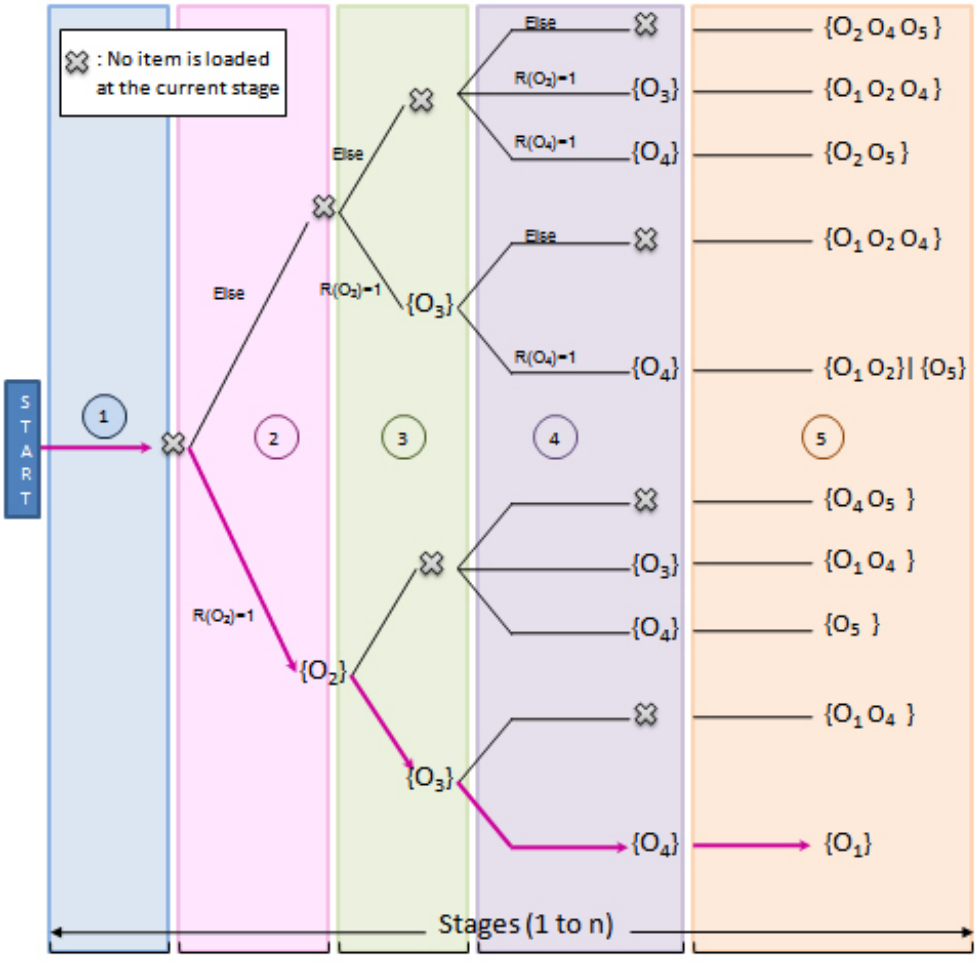


Figure 3.3: A decision strategy example for $n = 5$

We marked on the tree, with the pink color, the road of the optimal solution supplied by algorithm 3. We notice that all generated solutions are optimal; they provide the same profit even if the collection of items is different.

3.6 Conclusion

We proposed a solution approach for the DKP that incorporates a stopping rule at each stage of the loading process in which case the DM is able to make decisions throughout the searching process, for each incoming item. This approach was appealed to reduce resolution difficulties of the dynamic process. Using the OSP terminology, we determine the stage's stopping rule. A knapsack sub-problem is solved thereafter, given as input the set of candidates selected by mean of the stopping rule.

Chapter 4

Computational Experiments

4.1 Introduction

In the preceding chapter, we developed a new algorithm to solve the dynamic assignment of offers to a single and a constrained capacity resource. We adopted a dynamic formulation to compute, at each stage of the selection process, our gain if we choose the current offer and when delaying it. However, to make sure of our approach reliability, our algorithm shall be validated by real cases.
This chapter deals with the experimental investigation. Experimentation obtained by means of the proposed algorithm will be presented. We are looking to measure the effectiveness and the performance of our algorithm to produce a satisfiable solution. Therefore, we develop some metrics to show the effectiveness of our results.

We implemented both algorithms, discussed in the preceding chapter, in java language on a Intel Centrino Duo processor and 2GB of RAM under Microsoft Vista. We remind, the algorithms' basics: the first algorithm seeks for the optimal solution of the OSKP, while the second draws the correspondent decision strategy. Each algorithm is built for different instance sizes. For small n, we derive a solution strat-

egy for each knapsack, taking into account that each step may lead to a loading of some candidates or not, according to the attributed ranks. With large instances, generating a whole decision strategy is practically insignificant. Therefore, we derive just the optimal solution. A ranking method is used here to order items from the best to the worst, based on a computed ratio.
In both cases, we generate items coefficients randomly and uniformly within $\{1, 1000\}$. The knapsack capacity is then fixed to 50% of the sum of items weight.

The remainder of the present chapter is organized as follows. The second section will be concerned by the description of the experimentation's settings. We will differentiate thereafter between small and large instances. For each problem size, we present results of built instances and discuss the obtained values.

4.2 The experimental settings

Experiment results for the proposed algorithms are presented below. We have considered small and large instances. For small instances, varying in {5,35}, the algorithm generates a whole solution strategy considering the different manners to fill the same knapsack with the same objects, but in different order. At each stage, we calculate the expected utilities of the present items with all possible ranks and then we derive a variety of solutions at the end. With large instances, we derive optimal solutions for different problem sizes, belonging in {10,1000}. Item features (value and weight) are positive integer variables. To conduct experimentation, we generate those coefficients randomly and uniformly within the interval [1,1000]. The knapsack capacity was limited at a 50% of the sum of items weight.

4.3 Interpretation of the results

This section discusses the interpretation of the obtained results. We conduct two directions of experimentation: small instances for the DSA and large instances for the OSA. Note that both algorithm results were compared to that of a static algorithm, where items are assumed to be present simultaneously to the evaluation. Moreover, we present results comparison supplied by our OSA for different utility functions (U_1 and U_2).

4.3.1 Small instances

We consider first small instances for which we derive solution strategies. Instances were considered with n within the range {5,35} and different number of instances were performed for each problem size.

The overall results are presented in Table 4.1, where the minimum (Min), the average (Avg) and the maximum (Max) for the DSA are displayed. Table columns refer respectively to: the number of items (n), the number of instances (# instances), the Number of Possible Collections of items (NPC), the Average Reward (AR), the Percentage of Filling of the knapsack (PF)and the computational time (CPU).

Table 4.1: Experimental results for the DSA

n	**# instances**		**NPC**	**AR**	**PF(%)**	**CPU**
		Min	4	544	71.72	0.00011
5	15	Avr	7	1356	84.4	0.00017
		Max	8	2218	97.01	0.00025
		Min	48	1895	90.43	0.00075
10	10	Avg	57	2960	95.02	0.0011
		Max	68	3966	98.17	0.0021
		Min	277	4447	93	0.0027
15	5	Avg	388	4989	96.23	0.0072
		Max	495	6577	98.41	0.012
		Min	4459	4250	96.56	0.05
20	7	Avg	6972	5792	98.33	0.1136
		Max	7700	7544	99.02	0.14
		Min	51144	7078	98.51	0.39
25	7	Avg	58001	7893	98.89	0.82
		Max	64622	9120	99.20	1.08
		Min	592571	8105	98.57	5.9
30	8	Avg	905964	9773	99.12	19.98
		Max	1000045	11470	99.45	33.12
		Min	2809524	7346	99.12	42.5
33	4	Avg	3562328	9450	99.26	78.2
		Max	4007631	12213	99.52	101.75
		Min	7161272	10576	99.27	600
35	3	Avg	3711508	11127	99.29	671
		Max	7461745	10877	99.39	742.17

The Number of Possible Collections (NPC)

Our DSA generates various possibility of loading. Each possibility represent a collection of different items. Evidently, the NPC is growing with the problem size as shown in figure 4.1. However, the NPC is not equivalent to the mathematical combination because our algorithm generates only optimal solutions but from different valuation directions. Therefore, our algorithm supplied a variety of solutions and it is for the DM to decide the convenient.

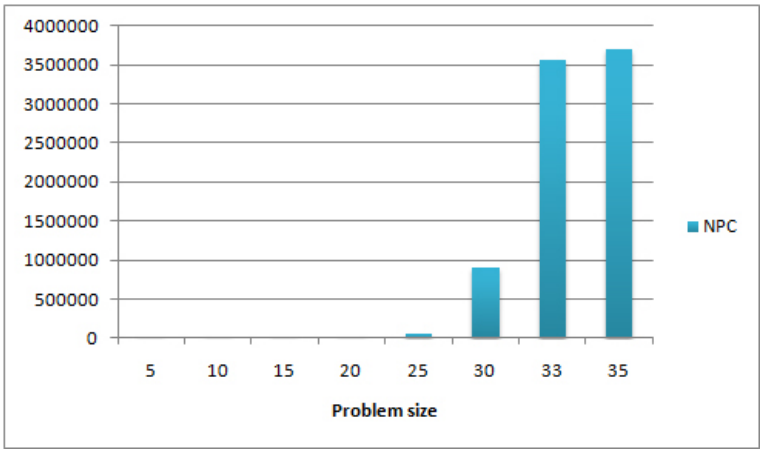


Figure 4.1: Progress of the number of possible collections of items in terms of problem sizes

The Average Reward (AR)

We present the average reward for built instances in order to compare it with static results. Figure 4.2 is a comparison histogram of our algorithm's AR and the AR of the same instances obtained statically. We notice that our results are about 80% of the optimal values.

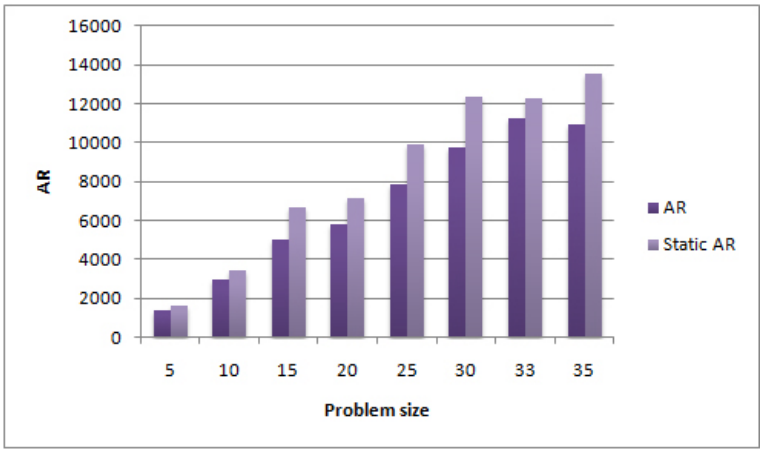


Figure 4.2: Effectiveness of the DSA results compared by those of a static algorithm

Percentage of Filling of the knapsack (PF)

Our second interest is to exhaust the capacity constraint; to fill the knapsack as much as possible. Therefore, our resolution approach has to take into consideration this objective too and this to avoid resources wasting.

The PF is given by:

$$PF = \frac{\sum_{i=1}^{m} w_i}{C} \times 100 \tag{4.1}$$

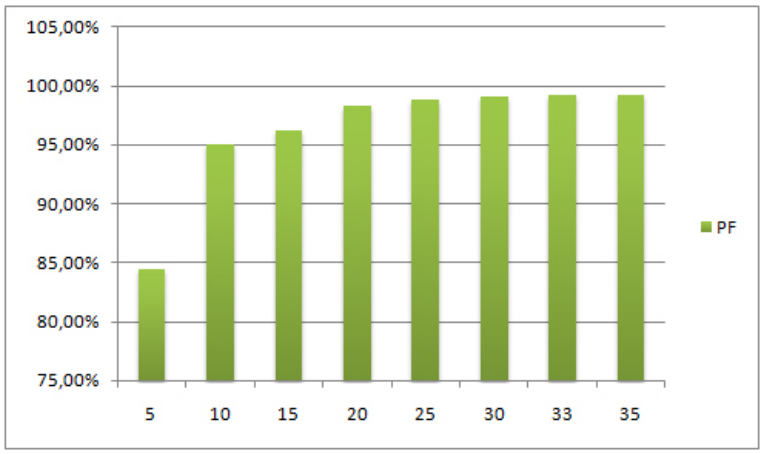


Figure 4.3: Evaluation of the percentage of filling of the knapsack for different problem sizes

Results shows that the PF is closest to 100%, especially for the greatest problem sizes. This indicates that the capacity constraint has been well exhausted and there are no loss of resources.

The CPU time

Figure 4.4 reports the progress of the computational time in terms of the problem size. The CPU rises relatively to the NPC too.

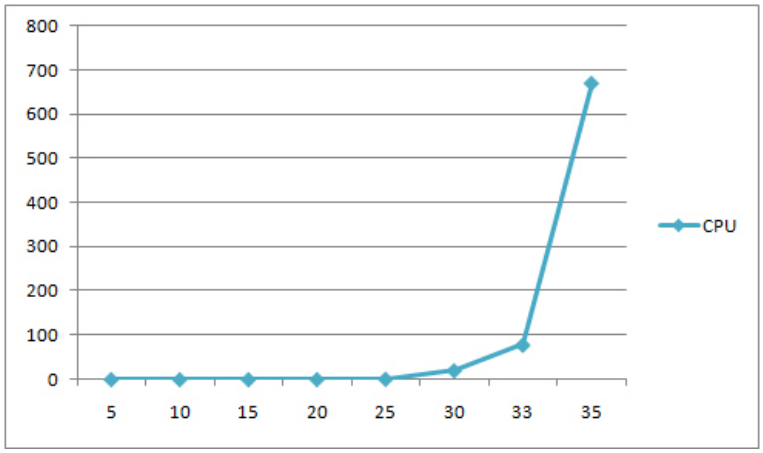


Figure 4.4: The CPU time behavior in terms of problem sizes

4.3.2 Large instances

Now we consider large instances, where the number of items n is within {10,1000}. For each problem size, 10 instances were performed. Table 4.2 reports results for the OSA in terms of a number of performance measures, which are respectively: the number of loaded items, the average reward, the percentage of filling of the knapsack, the first load stage, the percentage of loading before the last stage and the computational time. All instances has been performed for both utility functions presented in the earlier chapter.

Results showed that the NLI, AR and the CPU increase proportionally to the problem size. In what follows, we define the rest of used measures and explain this choice.

Table 4.2: Overall results for the OSA

n		NLI		AR		PF(%)		FLS		LBLS (%)		CPU
		U_1	U_2	U_1	U_2	U_1	U_2	U_1	U_2	U_1	U_2	
	Min	5	5	3507	3507	90.57	87.98	4	4	0.0	0.0	0.0001
10	Avg	6	6	3848	3825	96.05	95.49	7	7	23.14	33.57	0.0004
	Max	7	7	4202	4202	99.92	99.39	10	10	33.33	66.66	0.0014
	Min	30	30	17884	17884	99.53	99.35	17	14	3.032	21.87	0.001
50	Avg	32	32	20019	20019	99.74	99.74	28	23	7.38	28.29	0.017
	Max	33	33	23202	23202	99.95	99.95	43	39	12.5	33.33	0.002
	Min	60	60	39332	39332	99.91	99.91	33	31	1.61	18.03	0.003
100	Avg	62	62	41283	41283	99.96	99.96	68	58	3.72	23.9	0.011
	Max	65	65	42311	42311	99.99	99.99	97	85	6.15	30	0.015
	Min	121	121	78016	78016	99.92	99.92	93	56	0.81	24.0	0.14
200	Avg	124	124	81557	81557	99.97	99.97	143	93	3.62	26.43	0.15
	Max	127	127	84735	84735	100	100	198	172	5.78	29.75	0.17
	Min	183	184	118636	118636	99.98	99.98	122	88	0.0	19.78	0.514
300	Avg	190	190	123109	123110	99.994	99.993	228	169	1.9	24.11	0.74
	Max	197	197	127096	127096	100	100	300	215	3.14	25.38	0.8
	Min	243	243	154889	154889	99.97	99.97	133	107	0.4	21.82	1.66
400	Avg	251	251	161030	161031	99.98	99.98	260	142	2.08	24.71	2.24
	Max	257	257	166775	166775	99.99	99.99	373	184	3.57	29.62	2.65
	Min	309	308	191223	191223	99.98	99.98	223	138	0.63	22.72	2.11
500	Avg	314	314	202696	202696	99.99	99.99	353	268	1.78	24.84	5.08
	Max	320	320	214782	214782	100	100	496	389	2.57	26.25	9.41
	Min	369	369	235853	235853	99.995	99.995	210	160	1.89	21.72	4.91
600	Avg	374	374	243081	243081	99.998	99.998	324	267	2.25	24.38	9.08
	Max	383	383	249203	249203	100	100	585	501	3.39	27.49	13.01
	Min	427	427	271964	271964	99.992	99.992	236	201	1.15	22.95	12.77
700	Avg	436	436	282368	282368	99.997	99.997	450	320	1.83	25.65	13.95
	Max	445	445	296739	296739	100	100	678	480	2.76	27.79	17.6
	Min	489	489	314685	314685	99.996	99.996	267	213	1.43	22.49	13.31
800	Avg	501	501	326031	326031	99.998	99.998	476	402	1.73	24.97	22.05
	Max	516	516	332545	332545	100	100	787	737	2.13	25.58	29.5
	Min	552	552	358579	358579	99.996	99.996	312	244	0.9	22.82	34.22
900	Avg	563	563	366009	366009	99.998	99.998	685	605	1.27	24.59	36.93
	Max	576	576	369977	369977	100	100	894	871	1.73	25.34	40.79
	Min	615	615	400685	400685	99.996	99.996	340	266	0.48	24.06	47.62
1000	Avg	626	626	410631	410631	99.998	99.998	696	484	1.5	24.78	54.02
	Max	639	639	421587	421587	100	100	985	809	2.19	27.69	67.44

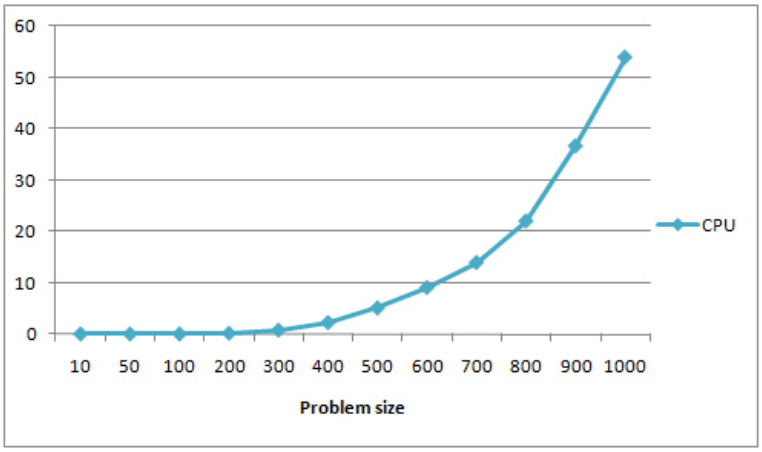


Figure 4.5: The CPU time progress for large n

Percentage of Filling of the knapsack (PF)

The PF is all the time close to 100%; the capacity constraint has been well exhausted and the knapsack is almost full.

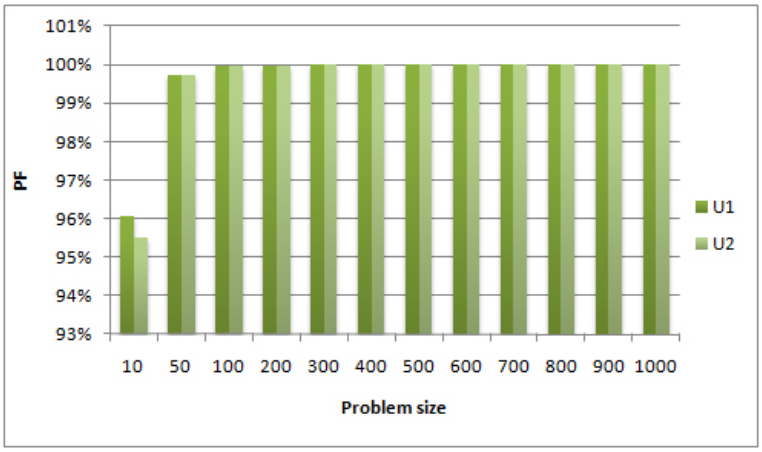


Figure 4.6: The effectiveness in filling the knapsack for both utility functions

The First Load Stage (FLS)

This measurement indicates at which stage of the process the algorithm began to load items. In other words, the stage in which the DM met a satisfiable offer.

Figure 4.7 draws the FLS position among the overall number of stages for each problem size, e.g. for static algorithms, the FLS is equivalent to the last stage that is first the load will be done at 100% of the searching process. Moreover, the figure compare the variation of the FLS according to the utility function.

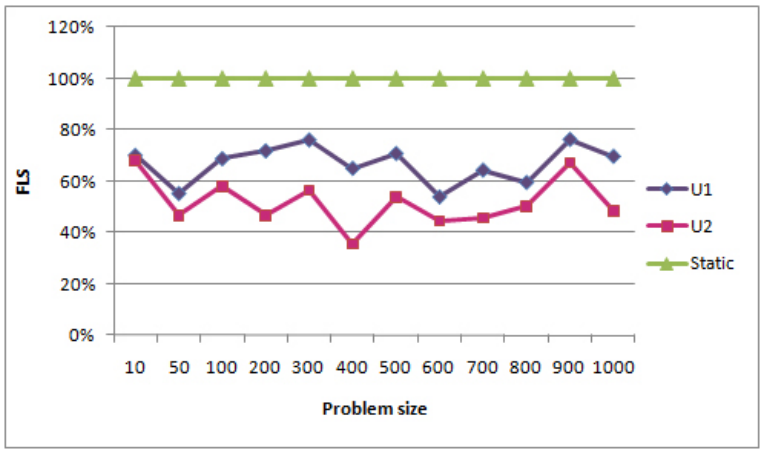


Figure 4.7: Variation of the first load stage by comparing U_1 and U_2

We notice that the FLS under the utility function U_1 is farther than that of U_2. Therefore, the utility function U_2 is more convenient for DMs who desires to make decisions in a close time horizon, while U_1 is more suitable for DMs that desires to delay their decisions until a considerable number of items appears.

The percentage of Loading Before the Last Stage (LBLS)

Throughout the selection process, we enumerate at each stage the number of loaded items before the last stage (before observing all items). We have considered this measure to learn about the efficiency of the dynamic approach in minimizing the loading process duration; thanks to the dynamic approach, we can begin to load items as soon as they arrive, we do not need to wait until all offers are received.

LBLS is written as follows:

$$LBLS = \frac{NLIBF}{NLI} \times 100 \tag{4.2}$$

NLIBF and NLI denote respectively: the number of loaded items before the final stage and the total number of loaded items.

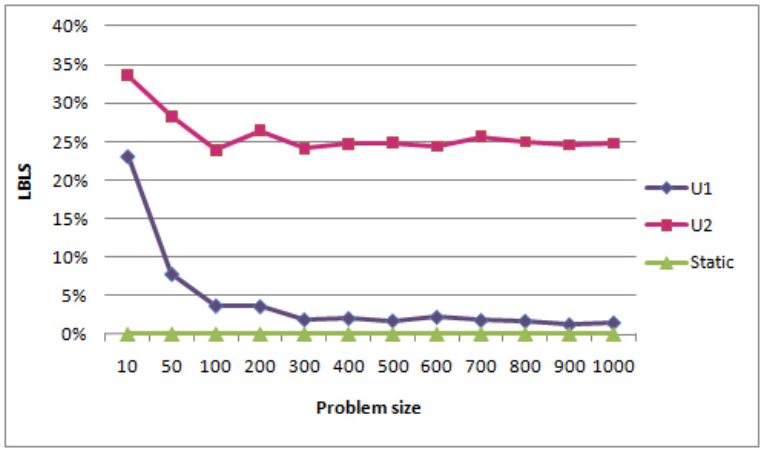


Figure 4.8: The LBLS behavior in terms of U_1 and U_2 compared to the static case

Figure 4.8 reports the LBLS behavior for each of the utility functions compared to the static case. The LBLS, given U_1, showed that while appealing to the dynamic loading, the final stage still of a considerable importance regarding the number of items selected at the end; the last stage monopolizes the biggest proportion of loads. However, the LBLS is improved using U_2: about 25% of items are loaded before the last stage. We notice that both curves have similar shapes but that of U_1 is lowest: this is due to the supported penalty, in term of utility, when an item is delayed. Indeed U_1 penalizes delayed items more than make it U_1.

To conclude, we can say that our algorithm has proved to be efficient in solving the DKP. Compared with static results, we reached almost the same overall profit. Besides to respecting the capacity constraint, we were able to fill at most the knapsack and making decision in opportune time. However, the utility function U_2 proved to be more convenient in terms of FLS and LBLS. It gives more interest in newcomers if compared with U_1, which prefer to delay as much as possible and makes decision

in latest stages. In the other hand, the utility function does not contribute in the overall reward nor in the PF; we reached all the time the same values using either function utility.

4.4 Conclusion

We presented in this chapter the empirical study. Our results showed the optimality in filling the knapsack for all problem sizes. This demonstrate the effectiveness of our approach in solving dynamically the DKP. However, we notice that a large proportion of loading is delayed to the last stage of the process. Therefore, further improvements can be brought to our formulation in order to maintain a satisfactory percentage of loading over stages.

General Conclusion

The OSKP is a combinatorial optimization problem that arises for numerous practical applications that involve an online scenario, and particulary for those of the DKP and the OSP. Moreover, our search was motivated by the fact that there are a modest number of research concerned by the DKP and that we found no research who evoked the possibility of delaying offers with a dynamic process.

Indeed the OSKP is a DKP involving a single DM observing a sequence of incoming offers over time, in order to select the optimal subset. No information is available about potential items before the arrival time, except of their total number. Offers are evaluated upon arrival and can be delayed to a subsequent stage or loaded immediately. However, once inserted into the knapsack, it cannot be removed to place another one. In order to load the best possible subset among all items, a careful decision should be taken on the acceptance or the rejection of each observed item. Therefore, we appealed to the OSP which provides a dynamic expectation of the utility of the current item.

We proposed a dynamic strategy for the DKP managed by an optimal stopping rule. The proposed approach controls the loading process stage by stage, evaluates each incoming item at the arrival time and attributes a new ranking of already observed items. Based on these ranks, we derived the item's utility value, required for the dynamic calculation. The stage's dynamic computation determines which items to

load in the knapsack and those should be delayed. At each stage of the dynamic process, items that can be packed are inserted into a static knapsack sub-problem solved using the simplex method.

We implemented the proposed approach to measure the effectiveness of the optimal stopping rule in such a problem. We designed two algorithms of the same principal; the first one determines the optimal filling subset while the second derives a decision strategy. Both algorithms were validated by the experimentation. Results showed the time savings and the optimality in filling the knapsack when using our dynamic approach.

One aspect that we would like to explore in the future is the extend of our model to more than a unique DM. Besides a possible generalization is to study the OSKP for possible unavailable items, where some of the delayed items get lost during the searching process.

Bibliography

Fouad Ben Abdelaziz and Saoussen Krichen. An interactive method for the optimal selection problem with two decision makers. *European Journal of Operational Research*, 162:602–609, 2005.

Gagan Aggarwal and Jason D. Hartline. Knapsack auctions. In *SODA '06: Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 1083–1092. ACM, 2006.

Moshe Babaioff, Nicole Immorlica, David Kempe, and Robert Kleinberg. A knapsack secretary problem with applications. In *APPROX '07/RANDOM '07: Proceedings of the 10th International Workshop on Approximation and the 11th International Workshop on Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 16–28, Berlin, Heidelberg, 2007. Springer-Verlag.

Cristina Bazgan, Hadrien Hugot, and Daniel Vanderpooten. Solving efficiently the 0-1 multi-objective knapsack problem. *Computers and Operations Research*, 36 (1):260–279, 2009.

J. Neil Bearden, Ryan O. Murphy, and Amnon Rapoport. A multi-attribute extension of the secretary problem: Theory and experiments. *Journal of Mathematical Psychology*, 49:410–422, 2005.

Young Hak Chun. Selecting the best choice in the weighted secretary problem. *European Journal of Operational Research*, 92:135–147, 1996.

Young Hak Chun. Optimal partitioning of groups in selecting the best choice. *Computers & Operations Research*, 28:1367–1386, 2001.

Ludwig Ensthaler and Thomas Giebe. Subsidies, Knapsack Auctions and Dantzig's Greedy Heuristic. *SSRN eLibrary*, 2009.

Thomas S. Ferguson. Who solved the secretary problem. *Statistical Science*, 4: 282–296, 1989.

Thomas S. Ferguson. Best-choice problems with dependent criteria. *Contemporary Mathematics*, 125:135–151, 1992.

Ben Abdelaziz Foued, Chaouachi Jouhaina, and Krichen Saoussen. Métaheuristiques de résolution du problème de sac à dos multiobjectif. In *Actes de la première conférence Francophone en Modélisation et Simulation des Systèmes de Production et de Logistique MOSIM'97*, pages 443–450. Editions Hermes, 2000.

Michael M. Guntzer and Dieter Jungnickel. Approximate minimization algorithms for the 0/1 knapsack and subset-sum problem. *Operations Research Letters*, 26: 55–66, 2000.

Kazuo Iwama and Shiro Taketomi. Removable online knapsack problems. In *ICALP '02: Proceedings of the 29th International Colloquium on Automata, Languages and Programming*, pages 293–305. Springer-Verlag, 2002.

George S. Lueker. Average-case analysis of off-line and on-line knapsack problems. In *SODA '95: Proceedings of the sixth annual ACM-SIAM symposium on Discrete algorithms*, pages 179–188. Society for Industrial and Applied Mathematics, 1995.

Mark Manasse, Lyle McGeoch, and Daniel Sleator. Competitive algorithms for on-line problems. In *STOC '88: Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 322–333. ACM, 1988.

A. Marchetti-Spaccamela and C. Vercellis. Stochastic on-line knapsack problems. *Mathematical Programming*, 68:73–104, 1995.

Silvano Martello and Paolo Toth. *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley and Sons, 1990.

David P. Morton and R. Kevin Wood. On a stochastic knapsack problem and generalizations. In D.L. Woodruff, editor, *Advances in Computational and Stochastic Optimization, Logic Programming and Heuristic Search: Interfaces in Computer Science and Operations Research*, pages 149–168. Kluwer Academic Publishers, 1998.

Jason D. Papastavrou and Anton J. Kleywegt. The dynamic and stochastic knapsack problem. *Operations Research*, 46:17–35, 1998.

Jason D. Papastavrou and Anton J. Kleywegt. The dynamic and stochastic knapsack problem with random sized items. *Operations Research*, 49:26–41, 2001.

Jason D. Papastavrou, Srikanth Rajagopalan, and Anton J. Kleywegt. The dynamic and stochastic knapsack problem with deadlines. *Management Science*, 42:1706–1718, 1996.

Ulrich Pferschy and Joachim Schauer. The knapsack problem with conflict graphs. *Journal of Graph Algorithms and Applications*, 13:233–249, 2009.

Zdzislaw Porosinski. On optimal choosing of one of the k best objects. *Statistics & Probability Letters*, 65(4):419–432, 2003.

Aminto Senisuka, Byungjun You, and Takeo Yamada. Reduction and exact algorithms for the disjunctively constrained knapsack problem. In *International Symposium on OR and Its Applications 2005*, Japan, 2005.

Nei Yoshihiro Soma and Paolo Toth. An exact algorithm for the subset sum problem. *European Journal of Operational Research*, 136:57–66, 2002.

Takeo Yamada, Seiji Kataoka, and Koutaro Watanabe. Heuristic and exact algorithms for the disjunctively constrained knapsack problem. *IPSJ Journal*, 43: 2864–2870, 2002.