Fatiha BENDALI, Jean MAILFERT Eloise MOLE-KAMGA, Alain QUILLIOT*, Hélène TOUSSAINT
LIMOS UCA/CNRS Laboratory, Labex IMOBS3
Clermont-Ferrand, FRANCE
(*) *: Corresponding Author*

A Synchronized *Feeder/Eater* Knapsack Problem.

I. Introduction.

Historically, Knapsack models [21, 30] arise as related to the problem of selecting among some set $I = \{1,\ldots, N\}$ of possibly identical items, those which are going to be packed into some *container* or *knapsack*, while meeting some capacity constraints and maximizing some economic utility value. Items may be clustered into classes or provided with characteristics, and this may induce additional constraints: exclusion constraints (pick up no more than k items with a same characteristic) or inclusion constraints (pick up at least p items with a same characteristic). Also, the container may impose several kind of capacity constraints, related for instance to the weight of the items or to their volume. In such a case, one talks about *multi-dimensional* Knapsack [15, 28, 29, 36]. According to this original interpretation, the order according to which items are selected does not matter and neither does the way items are indexed from 1 to N. A dual Knapsack formulation will involve items $i = 1,\ldots, N$ which have to be picked up in order to achieve at least some utility level, while minimizing some cost value.

Even when only one capacity constraint and no exclusion/inclusion constraints are imposed, those Knapsack problems are NP-Hard [34], mainly because of the encoding size of the numbers (weight and values) which are involved into the model. Still, in case those numbers are arbitrarily bounded and no exclusion/inclusion constraints exist, the problem may be solved in polynomial time. This opens the way to *Fully Polynomial Approximation Scheme* (FPTAS), which also hold if the number of dimensions of the model is fixed [8].

One of the best known extensions of basic Knapsack involves the possibility of letting several players act either on the variables (outputs) of the model or on both its inputs (parameters) and its outputs (variables): this leads to *bi-level* or *multi-level* Knapsack formulations [2, 6, 11]. Those players may be independent and driven by antagonistic purposes, and so resulting problem must be cast into the Game Theory (Stackelberg games, Non Cooperative games,…) framework, with the purpose of computing some kind of equilibrium (Core, Nucleolus, Nash equilibrium). They may also behave according to some kind of hierarchical relation, allowing the possibility of multi-objective reformulations [12]. Another extension involves the possibility to iterate several times the resolution of the problem, while taking into account some contextual changes (modifications of the input values,…), giving rise to *multi-stage* Knapsack models [3].

Still, in practice, it happens that many Knapsack models derive from the study of some dynamical system, and then the way decision is taken with respect to the items $i = 1,\ldots, N$ becomes sequential. According to such a context, items may be viewed as related to periods or steps in the achievement of some global process, and decision variables x_i, $i = 1,\ldots, N$, mean the activation of some control imposed to the system. This control may consist into a change of state, the consumption of some resource, the achievement of some job or the production of some good. Then a Knapsack solution becomes some kind of trajectory, which may be completely deterministic, or subject to some level of uncertainty that one usually tries to cast into the stochastic process framework [27]. This point of view makes Knapsack models get closer to some scheduling or resource allocation models [39], like for instance the *Lot Sizing* one [19]. It also provides a natural justification to the fact that those models, which are most often NP-Hard, are very often addressed through dynamic

programming techniques [6, 16, 18, 19], which in turn allow the formulation of pseudo-time-polynomiality statements [7, 9, 14], together with the design of *Polynomial Time Approximation Schemes* (PTAS) [35, 42].

This interpretation of Knapsack models as related to the control of dynamic systems or processes also provides multi-level and multi-stage extensions with very natural motivations: Multi-level Knapsack may involve several players acting on a same Knapsack trajectory according to divergent purpose [38, 43]; But it may also occur that those players are in charge of their own trajectory, and that those trajectories must interact because of a kind of common goal, as it may for instance happen when some sub-contractor mechanism links together the different players. This point of view prevails for instance when it comes to the management of distributed production systems, where encapsulated Lot Sizing [10, 13, 20] or Power management [4, 5, 17, 23] models aim at scheduling production units located at different levels of a same supply chain in order to meet a final end-user predetermined or uncertain demand [22]. However, we notice that in such a case, there exists a clear distinction between the players involved at the different levels or stages of the production process, and this final end-user. This allows managers to set global models driven by some global producer purpose, and to deal with them according to the centralized decision paradigm.

Now it may happen that this distinction cannot be done, and that the different levels of the problem express some kind of *producer-consumer* interaction between players which must synchronize themselves because of capacity constraints. We may for instance consider the case when one of the players is a local solar energy producer *Feeder* [1, 17], which produces solar energy (for instance solar hydrogen) all along a time space divided into N periods $i = 1,\ldots, N$, and when the other(s) player(s) is (are) an industrial or service player(s) *Eater*, which needs this energy in order to achieve some process. In such a case this *eater* player will visit some stations $j = 1,\ldots, M$, and, every time it arrives to station j, it will decide whether it performs or not some job related to j. If *Feeder* is provided with production prevision which makes correspond, to any period i, a production P_i according to cost C_i, then it should schedule its production periods as a {0, 1}-vector $z = (z_i, i = 1,\ldots, N)$, solution of the following *Feeder Knapsack* model:

- $\Sigma_i P_i.z_i \geq Cons$, with *Cons* = the total energy amount that *Eater* is going to require,
- $\Sigma_i C_i.z_i$ minimal.

On the other side, if performing job j requires S_j energy units, if every job j is provided with some utility value W_j, and if we impose that some global utility value W^* must be achieved by the *Eater*, *Eater*'s problem is to decide about a {0, 1}-vector $x = (x_j, j = 1,\ldots, M)$ which is going to tell it which jobs have to be performed and which will be solution of the following *Eater Knapsack* model:

- $\Sigma_j W_j.x_j \geq W^*$,
- $\Sigma_j S_j.x_j = Cons$.

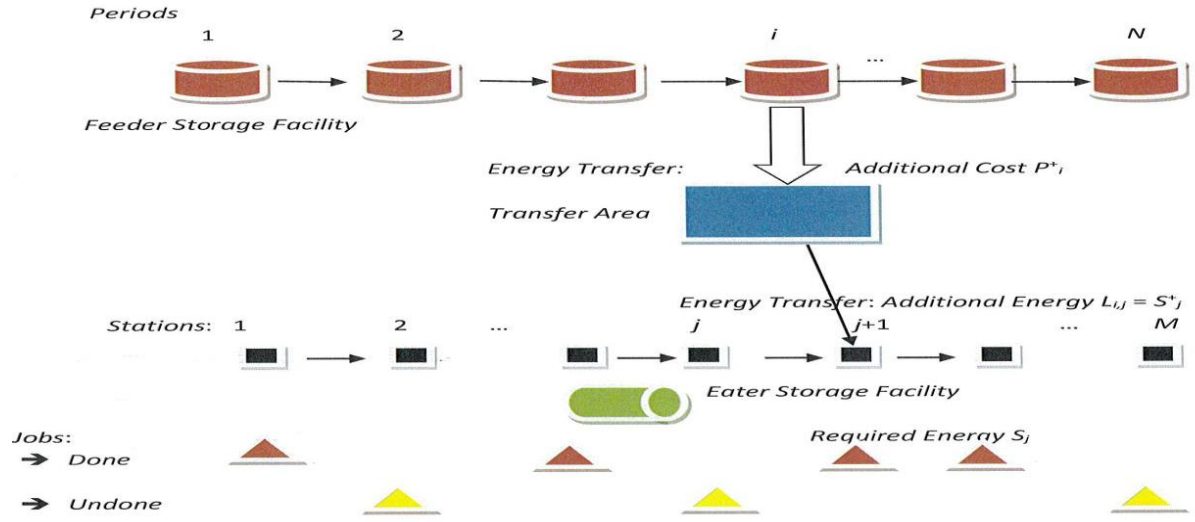

Figure 1: *Energy Feeder/Eater Interaction Process*.

In order to implement such a process, *Feeder* and *Eater* have to agree about periods i and stations j when energy transfers take place and about related transferred energy amounts $L_{i,j}$. Let us suppose that such a *transfer transaction* (i, j, $L_{i,j}$) takes place between period i and period i +1 and requires *Eater* to spend S^+_j additional energy units while also inducing an additional transfer cost C^+_i for *Feeder*. Then, if vector $u = (u_i, i = 1,\ldots, N)$ means the periods related to those transfer transactions, and if $v = (v_j, j = 1,\ldots, M)$ means related stations, then above requirements are updated as follows:

- $\sum_i C_i.z_i$ is replaced by $\sum_i C_i.z_i + \sum_i C^+_i.u_i$ and we get that $\sum_i C_i.z_i + \sum_i C^+_i.u_i$ must be minimized;
- $\sum_j S_j.x_j$ is replaced by $\sum_j S_j.x_j + \sum_j S^+_j.v_j$ which must be equal to $Cons = \sum_{i,j} L_{i,j}$.

Now we have to take into account that in most cases, both *feeder* and *eater* players have to meet capacity constraints. For instance, if we refer to above example related to solar energy, *Eater* may store this energy inside a storage facility (battery, tank,…) with limited capacity C^{Eat}. A consequence is that it will have to decompose its decision process x into a sequence of encapsulated Knapsack stages, delimited by the stations j such that $v_j = 1$: Between 2 such delimiting values j_1 and j_2, *Eater* will not be able to perform jobs globally requiring more energy than the amount it was provided with after a transfer occurred in j_1.

The problem described this way may be viewed as a kind of bi-level Knapsack problem with flexible stages. Main issue is about the synchronization of *feeder* and *eater* processes through transfer transactions (i, j, $L_{i,j}$).. It may arise in various contexts, some related to energy management [31, 33, 37], and other not: Real time cooperation between sensors provided with limited memory, and a robot control process which decides about trajectory according to signals and patterns sent by the sensors [32, 40]; Cooperation between a *feeder* manager in charge of the maintenance of some equipments, which decides about re-hauling periods and levels, and an *eater* equipment manager, which schedules the use of the equipments according to its own purposes and to the current state of the equipments [33]; Situations involving asset management [25, 41]; Electric vehicle routing with battery recharging [24, 26, 37], … . Depending on the context, resulting models may differ in their details: for instance, if we refer to the energy context, a temporal dimension will be in most cases part of the model, related to both the duration of the production periods 1,…, N and the processing times of jobs 1,…, M; Also, if indices 1,…, M are related to specific locations where the jobs may be achieved, not performing any job in j may impose the *eater* player to skip station j, and induce a constraint $v_j \leq x_j$. Still, the core of the problem will remain, which is about the way the different decision levels interact and synchronize themselves at every stage of the global *Feeder/Eater* process.

Above discussion makes appear that, depending on the context, multi-level does not clearly identify here a *leader* and a *follower*. As a matter of fact, in true life, the players involved into such a *feeder/eater* process are most often, at least to some extent, independent and driven by their own agenda. This raises the *centralized versus collaborative* issue, and part of the challenge becomes then designing models and algorithmic tools which might help them in collaborating.

So the problem which we are going to study here, which we call **SFEK** (*Synchronized Feeder/Eater _Knapsack*) Problem, refers to the issues we have just been discussing. It is defined by 2 Knapsack models which are linked together by the kind of *transfer* mechanism which we have been describing in our example related to the management of renewable energy. Though the *centralized versus collaborative* issue is very important in practice, we shall hardly address it here, and conform ourselves to the centralized paradigm: That means that we shall presuppose the existence of a central decider or mediator which has authority on the system and so may impose a solution suggested by some computational model.

We shall first (Section II) provide a formal description of the basic **SFEK** problem. In the same section, we shall discuss the way this model may be adapted or extended in order to fit a given context, which may for instance involve multi-objective features or a temporal dimension. In Section III, we shall propose a MILP (*Mixed Integer*

Linear Programming) formulation of **SFEK**, and discuss the way additional valid constraints as well as a lower bound may be obtained, which might make easier the handling of this problem by some MILP software. Section IV will be devoted to our main contribution, namely a dynamic programming algorithm which will allow us to derive a polynomial-time approximation scheme (PTAS) for **SFEK**. Finally, we shall conclude with a brief discussion about a decomposition scheme which might open the way to a decentralized decision point of view and the design of a collaborative approach.

II. The *SFEK*: *Synchronized Feeder/Eater Knapsack* Problem.

We are going to set here the **SFEK** Problem while implicitly referring to the example previously described, related to the management of renewable energy. That means that we are going to deal with 2 *Knapsack players*, one called the *feeder player* (the energy producer), the other one being the *eater player*. Both will be supposed to take decisions in a sequential way and communicate through *transfer transactions* from the *feeder* to the *eater*, while being submitted to capacity constraints.

So we consider:

- An index set $J = \{1,\ldots, M\}$, which we call *eater station space*, related to the *eater player*, together with:
  - A *resource* vector $S = (S_j, j = 1,\ldots, M) \geq 0$ and an *additional resource* vector $S^+ = (S^+_j, j = 1,\ldots, M) \geq 0$;
  - A *utility* vector $W = (W_j, j = 1,\ldots, M) \geq 0$ together with a *utility* threshold W^*;
  - A *eater capacity* coefficient C^{Eat};
  - An *initial resource* coefficient H^{Eat}.
- An index set $I = \{1,\ldots, N\}$, which we call *feeder period space* related to a *feeder player*, together with:
  - A *resource* vector $P = (P_i, i = 1,\ldots, N) \geq 0$;
  - A *cost* vector $C = (C_i, i = 1,\ldots, N) \geq 0$ and an *additional cost* vector $C^+ = (C^+_i, i = 1,\ldots, N) \geq 0$;
  - A *feeder capacity* coefficient C^{Feed};
  - An *initial resource* coefficient H^{Feed}.

Related decision system works as follows:

- The *eater* player visits stations of $J = \{1,\ldots, M\}$ in this order and, for any station j, it takes a *station* decision x_j (to perform some job related to j): If $x_j = 1$ then it spends S_j resources and gets a utility value W_j, while the current level of its available resources is decreased by S_j. This level is initially H^{Eat}, and it must always be located between 0 and C^{Eat}. At the end of the process, the *eater* must have achieved a global utility value at least equal to W^*.
- In order to provide the *eater* player with resources, the *feeder* player takes, at any period i, a *period* decision z_i: $z_i = 1$ means that some amount P_i of resources is produced at a cost C_i, and the current level of *feeder*'s available resources is increased by P_i. This level is initially equal to H^{Feed} and must always be located between 0 and C^{Feed}.
- Both *feeder* and *eater* players interact through *transfer transactions*: A *transfer transaction* T is a 3-uple $T = (i, j, L)$ whose meaning is: At the junction of periods i and $i + 1$, the *feeder* transfers L resource units to the *eater*, which receives them when it moves from station j to station $j + 1$. We say that i is the *origin* of T and j its *destination*. This transfer transaction requires an additional cost C^+_i from the *feeder* and an additional resource amount S^+_j from the *eater*. The current level of available resources of the *feeder (eater)* player decreases (increases) accordingly. In order to fit with the idea that decisions are taken sequentially, we impose that if 2 transactions are performed, respectively related to i_1, j_1 and i_2, j_2, then $j_1 < j_2$ iff $i_1 < i_2$. (E1:*Transfer Consistency* Constraint)
  It comes that a *transfer schedule* is defined as a collection *Trans* of transfer transactions, which satisfies above *Transfer Consistency* constraint. We see that we may derive from such a transfer schedule a *global transfer amount* $L^* = \sum_{T \in Trans} L$, two {0, 1}-valued vectors $u = (u_i, i = 1,\ldots, N)$ and $v =$

(v_j, j = 1,…, M), together with non negative vectors L^{Feed} = ($L^{Feed}{}_i$, i = 1,…, N) and L^{Eat} = ($L^{Eat}{}_j$, j = 1,…, M), whose meaning is:

- u_i = 1 iff there exists a transfer transaction T = (i, j, L) = (i, j, $L^{Feed}{}_i$) with origin i in *Trans*; If u_i = 0 then $L^{Feed}{}_i$ = 0; We must have $\Sigma_i L^{Feed}{}_i = L^*$;
- v_j = 1 iff there exists a transfer transaction T = (i, j, L) = (i, j, $L^{Eat}{}_i$j with destination j in *Trans*; If v_j = 1 then $L^{Eat}{}_j$ = 0; We must have $\Sigma_j L^{Eat}{}_j = L^*$.

Then, introducing a fictitious period 0 allows us to describe the evolution of the level $V^{Feed}{}_i$ of available resources for the *feeder* at the end of any period i:

- $V^{Feed}{}_0 = H^{Feed}$ and for any $i \geq 1$, $V^{Feed}{}_i = V^{Feed}{}_{i-1} + z_i.P_i - u_i.L^{Feed}{}_i$. (E2)

By the same way, introducing a fictitious period 0 allows us to describe the evolution of the level $V^{Eat}{}_i$ of available resources for the *eater* when it leaves station j:

- $V^{Eat}{}_0 = H^{Eat}$ and for any $j \geq 1$, $V^{Eat}{}_j = V^{Eat}{}_{j-1} - x_j.S_j + v_j.(L^{Eat}{}_j - S^+{}_j)$. (E3)

This gives rise to the following **SFEK**: ***Synchronized Feeder/Eater Knapsack Problem***:

{**SFEK**: ***Synchronized Feeder/Eater Knapsack Problem***:
Compute *period* decision vector z = (z_i, i = 1,…, N), *station* decision vector x = (x_j,j = 1,…, M), together with transfer schedule *Trans* and global transfer value L^* in such a way that:

- Vectors x and v are a feasible solution of the following **Eater Knapsack** constraints:
  - $\Sigma_j x_j.W_j \geq W^*$;
  - $\Sigma_j x_j.S_j + \Sigma_j v_j.S^+{}_j \leq L^*$.
- *Feeder* and *eater* available resource levels $V^{Feed}{}_i$ and $V^{Eat}{}_j$ respectively satisfy:
  - (E2) and (E3); (*Evolution Constraints*)
  - $V^{Feed}{}_N \geq H^{Feed}$ and $V^{Eat}{}_M \geq H^{Eat}$; (*Initial Conditions*)
  - For any period i, $0 \leq V^{Feed}{}_i \leq C^{Feed}$; (*Feeder Capacity Constraints*)
  - For any station j, $0 \leq V^{Eat}{}_j \leq C^{Eat}$; (*Eater Capacity Constraints*)
- Vectors z meets the following **Feeder Knapsack** constraints:
  - $\Sigma_i z_i.P_i \geq L^*$;
  - The quantity $C.z + C^+.u = \Sigma_i (z_i.C_i + u_i C^+{}_i)$ is minimal}.

II.1. An Example.

Let us set M = 5, N = 7, W^* = 12, C^{Eat}= 6, C^{Feed}= 10, H^{Eat}= 3, H^{Feed} = 2, and vectors S, S^+, W, and P, C, C^+ as follows:

j	1	2	3	4	5
S	2	4	1	2	3
S+	2	1	3	1	2
W	3	4	2	1	4

7 8

i	1	2	3	4	5	6	7
P	3	2	4	1	4	2	5
C	4	1	5	2	6	1	8
C+	3	1	4	2	1	5	2

Then we get a feasible solution by setting: (transfer transactions are represented above with arrows)

- *Trans* = {(2, 1, 7), (6, 3, 8)}: At period 2, *Feeder* transfers 7 resource units to *Eater* located at station 1, and, at period 5, it transfers 8 resource units to *Eater* located at station 3,
- z = (1, 1, 1, 1, 1, 1, 0); u = (0, 1, 0, 0, 1, 0, 0); v = (1, 0, 1, 0, 0); x = (1, 1, 1, 0, 1).
- Related cost $\Sigma_i (z_i.C_i + u_i C^+{}_i)$ is equal to 21; Related value $W = \Sigma_j x_j.W_j$ is equal to 13.

The process may be illustrated by the following figures 2 and 3:

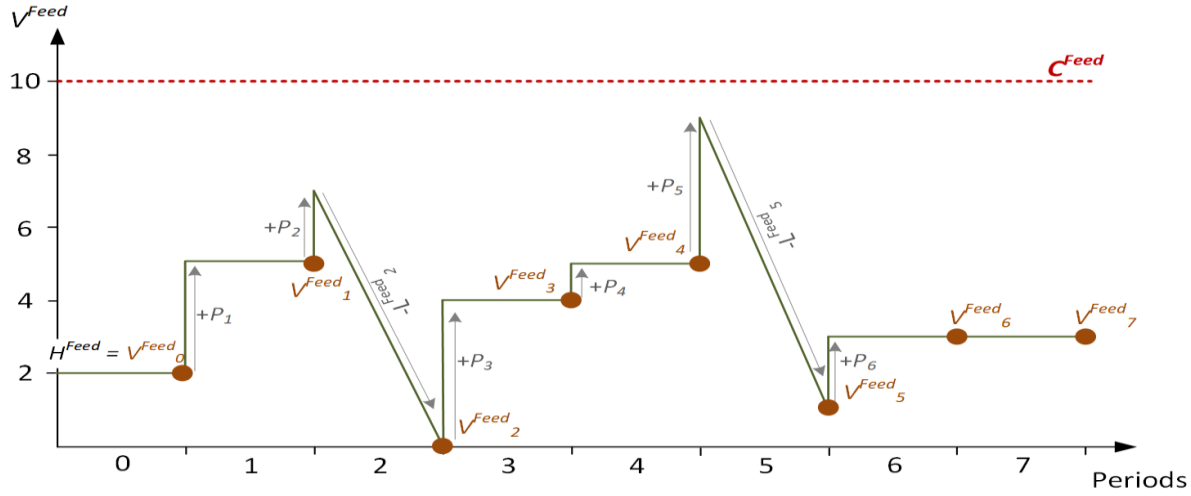

Figure 2: *Evolution of the feeder's resources as a function of the feeder periods*.

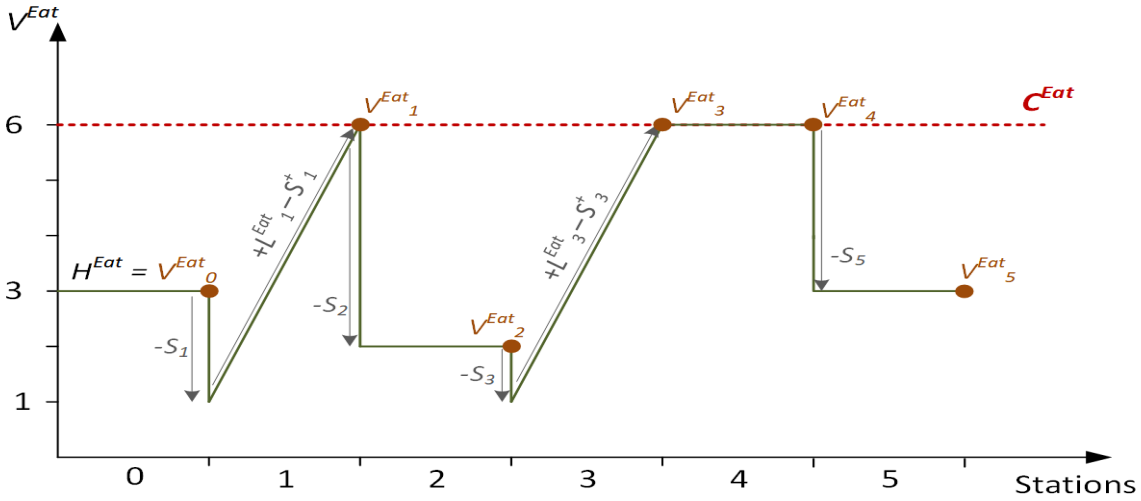

Figure 3: *Evolution of the eater's resources as a function of the eater stations*.

II.2. Possible Variants and Extensions of SFEK: Discussion.

Depending on the application context, above model may have to be adapted in several ways.

Variants related to the way transfer transactions are performed: It may happen that a transfer transaction cannot be viewed as instantaneous and must be considered as requiring one or more periods. In such a case, decision z_i must usually be null for any such a period, which also means the constraint $z_i + u_i \leq 1$. By the same way, jobs and stations may be the same, and so visiting stations means performing related job: A consequence is that in such a case, variable x_j and v_j must such that $x_j \geq v_j$. Also, the way formula (E2, E3) have to be may differ according to the context: For instance, a transfer at period i may be explicitly imposed to occur after the possible production of P_i, which will leads to a reinforcement of (E2) with the constraint (E2'): $V^{Feed}_{i-1} + z_i.P_i \leq C^{Feed}$. By the same way, a transfer at station j may be explicitly imposed to occur after the possible consumption of $x_j.S_j + v_j.S^+_j$, which leads to a reinforcement of (E3) with the following constraint (E3'): $V^{Eat}_{j-1} - x_j.S_j - v_j.S^+_j \geq 0$.

Multi-Objective SFEK: As told in the introduction, the *Feeder/Eater* interaction very often underlies situations where both Knapsack players are, at least to some extent, independent and driven by their own agenda. In such a case, we may have to distinguish the criterion $C.z + C^+.u$, which may be viewed as only expressing the

feeder player's concerns, from a criterion specific to the *eater* player, for instance the value $W.x = \Sigma_j W_j.x_j$. In such a case, we get a multi-objective version of our problem by replacing the objective:

Minimize $C.z + C^+.u$,

by the bi-objective:

Minimize both $C.z + C^+.u$ and $W.x$.

We possibly may simplify this bi-objective formulation by introducing a scaling coefficient α, and considering a unique criterion $C.z + C^+.u + \alpha.W.x$.

SFEK with Temporal Dimension: The way we motivated **SFEK** suggests that most applications are going to be related to scheduling and planning, and so involve a temporal dimension. This temporal dimension may usually be expressed as follows:

- Every *feeder period i* is provided with a non negative *duration* Δ_i; Any transfer transaction (*i*, *j*, *L*) involving *i* must take place at time $\Delta^*_i = \Sigma_{k \le i} \Delta_k$, which means at the end of period *i*.
- Moving from a station *j* to its successor (or visiting station *j*) requires a time amount t_j from the *eater* player, and performing related job requires additional time amount t^+_j. A transfer transaction (*i*, *j*, *L*) requires a time amount $\delta_{i,j}$ from the *eater* player: this means that if T_{j-1} is the time when the *eater* player ends with station *j*-1, then T_j is given by the formulas: (E4)
  - $T_0 = 0$;
  - If no transfer transaction involves *j*, then $T_j = T_{j-1} + t_j + x_j . t^+_j$;
  - If a transfer transaction (*i*, *j*, *L*) is performed which involves *j*, then we must have:

$$T_j = \Delta_i \ge T_{j-1} + t_j + x_j . t^+_j + \delta_{i,j} .$$

Then solving **SFEKT**: *SFEK with Time Dimension*, consists in computing vectors *z, x*, *Trans* as in standard **SFEK**, together with time values T_j, in such a way that both standard **SFEK** constraints and (E4) be satisfied, and that a mixed cost $C.z + C^+.u + \beta.T_M$ be the smallest possible, β being a scaling coefficient.

Example: We keep on with the example 1 of Section II.1, while adding following vectors Δ, *t*, t^+ and δ such that $\delta_{i,j}$ is independent on *i*.

j	1	2	3	4	5
S	2	4	1	2	3
***S*⁺**	2	1	3	1	2
W	3	4	2	1	4
t	1	1	2	0	1
***t*⁺**	6	4	5	4	2
δ	2	4	1	3	6

↑ 7 ↗ 8

i	1	2	3	4	5	6	7
P	3	2	4	1	4	2	5
C	4	1	5	2	6	1	8
***C*⁺**	3	1	4	2	1	5	2
Δ	6	4	2	8	5	6	4

Then we keep a feasible solution by setting:

- *Trans* = {(2, 1, 7), (5, 3, 8)};
- *z* = (1, 1, 1, 1, 1, 1, 0); *u* = (0, 1, 0, 0, 1, 0, 0); *v* = (1, 0, 1, 0, 0); *x* = (1, 1, 1, 0, 1).
- Related cost $\Sigma_i (z_i.C_i + u_i C^+_i)$ is equal to 21; Related value $W = \Sigma_j x_j.W_j$ is equal to 13.
- *T* = (10, 15, 25, 29, 32): Notice that the fact that transfer transaction (2, 1, 7) must take place at the end of period 2 imposes the *Eater* to wait before receiving resource amount 7. This remark holds for the transfer transaction (5, 3, 8).

Still, we see that if we set $t_2 = 4$ instead of $t_2 = 1$, then above solution becomes unfeasible, since the *eater* player does not have enough time in order to move through stations 2 and 3 while performing both jobs 2 and 3 between the time interval defined by periods 3, 4, 5.

During the next sections, we shall restrict ourselves to the study of the original **SFEK** model. Still, all results which are going to describe can be extended to the different **SFEK** variants which we just introduced.

III. A MILP Formulation of SFEK.

We first state (Complexity of **SFEK)**:

Proposition 1: ***SFEK*** *is NP-Hard*.

Proof: It comes in a straightforward way from the fact that **SFEK** contains 2 Knapsack models which interact. More precisely, we only need to set: C^+ = Null vector, S^+ = Null vector, $W^* = \Sigma_j W_j$, $S^* = \Sigma_j S_j$, $H^{Feed} = H^{Eat}$ = Large number, $C^{Feed} = C^{Eat} = +\infty$, in order to turn ***SFEK*** into a Standard Knapsack instance:

{Compute {0, 1} valued vector z such that $P.z \geq S^*$ and which minimizes $C.z$}.

We conclude. □

Though our purpose here is not to deal with MILP models, we are going to devote this section to the presentation of such a model and the way it may be enhanced through valid cuts. By this way, we shall remove any ambiguity from the way **SFEK** must be understood. In order to do it, we introduce the following vectors:

- ***Eater*** **variables**:
  - $x = (x_j, j = 1,\ldots, M)$, with {0, 1} values: $x_j = 1$ means that the *eater player* will perform the job related to station j;
  - $u = (u_j, j = 1,\ldots, M)$, with {0, 1} values: $u_j = 1$ means that *eater* will receive resource from the *feeder at station j*;
  - $V^{Eat} = (V^{Eat}_j, j = 0,\ldots, M) \geq 0$: V^{Eat}_j means the amount of resource available to the *eater* when it leaves station j; $j = 0$ is a fictitious period related to initialization;
  - $L^{Eat} = (L^{Eat}_i, j = 1,\ldots, M) \geq 0$: L^{Eat}_j means the resource transfer amount at station j;

- ***Feeder*** **variables**:
  - $z = (z_i, i = 1,\ldots, N)$, with {0, 1} values: $z_i = 1$ means that the *feeder* player decides to generate resources at period i;
  - $V^{Feed} = (V^{Feed}_i, i = 0,\ldots, N) \geq 0$: V^{Feed}_i means the amount of resource available to the *feeder* at period i; $i = 0$ is a fictitious period related to initialization;
  - $u = (u_i, i = 1,\ldots, N)$, with {0, 1} values: $u_i = 1$ means that the *feeder* will transfer resource to the *eater* at period i;
  - $L^{Feed} = (L^{Feed}_i, i = 1,\ldots, N) \geq 0$: L^{Feed}_i means the resource transfer amount at period i;

- ***Synchronization*** **variables**:
  - $U = (U_{i,j}, i = 1,\ldots, N, j = 1,\ldots, M)$ with {0, 1} values: $U_{i,j} = 1$ means that a *transfer* transaction (i, j, $L_{i,j} = L^{Feed}_i = L^{Eat}_j$ is performed at *station j* and *period i*.

Then our ILP formulation comes as follows:

***SFEK_MILP* MILP Formulation**:

{Compute vectors z, V^{Feed}, u, L^{Feed}, x, V^{Eat}, v, L^{Feed}, U in such a way that:

- **Objective function**: Minimize $\Sigma_{i=1,\dots,N}\ (C_i..z_i + C^{+}{}_i..u_i)$
- ***Feeder* constraints**:
  - $V^{Feed}{}_0 = H^{Feed}$; $V^{Feed}{}_N \geq H^{Feed}$;
  - For any $i \geq 1$,
    - $V^{Feed}{}_i = V^{Feed}{}_{i-1} + z_i.P_i - L^{Feed}{}_i$; (E5)
    - $C^{Feed} \geq V^{Feed}{}_i \geq 0$;
    - $L^{Feed}{}_i \leq u_i.\ C^{Feed}$;
- ***Eater* Constraints**:
  - $V^{Eat}{}_0 = H^{Eat}$; $V^{Eat}{}_M \geq H^{Eat}$;
  - $\Sigma_j\ W_j.x_j \geq W^*$;
  - For any $j \geq 1$,
    - $C^{Eat} \geq V^{Eat}{}_j \geq 0$;
    - $L^{Eat}{}_j \leq v_j.\ C^{Eat}$;
    - $V^{Eat}{}_j = V^{Eat}{}_{j-1} - x_j.S_j - v_j.S^{+}{}_j + L^{Eat}{}_j$; (E6)
- ***Synchronization* constraints**:
  - For any $j = 1,\dots, M$, $\Sigma_{i=1,\dots,N}\ U_{i,j} = v_j$; (E7_1)
  - For any $i = 1,\dots, N$, $\Sigma_{j=1,\dots,M}\ U_{i,j} = u_i$; (E7_2)
  - For any $j = 1, \dots, M$, $i = 1,\dots, N$: $U_{i,j} + (L^{Feed}{}_i - L^{Eat}{}_j)/\mathrm{Inf}(C^{Eat}, C^{Feed}) \leq 1$; (E8_1)
  - For any $j = 1, \dots, M$, $i = 1,\dots, N$: $U_{i,j} + (L^{Eat}{}_j - L^{Feed}{}_i)/\mathrm{Inf}(C^{Eat}, C^{Feed}) \leq 1$; (E8_2)
  - For any i, j, i_1, j_1 such that $i < i_1$ and $j_1 < j$, $U_{i,j} + U_{i1,j1} \leq 1$. (E9)

We may state:

Theorem 1: *Solving **SFEK_MILP** also solves the **SFEP**. Besides,* if C^{+} is the null vector and if $S_j + S^{+}{}_j/2 \leq C^{Eat}.(2N - 1)/2N$ for any $j = 1,\dots, M$, *then the optimal value of the rational relaxation of **SFEK_MILP** is null*.

Proof: Equations (E7) impose a one-to-one correspondence between the transfer transactions as viewed from the *feeder*'s point of view and the same transfer transactions viewed from the *eater*'s point of view. Then exclusion constraints (E9) ensure that related transfer schedule *Trans* will satisfy the consistency constraints (E1), while (E8), which is the linearization of the implication ($U_{i,j} = 1$) -> ($L^{Feed}{}_i = L^{Eat}{}_j$), makes that for any transfer transaction (i, j, L) in *Trans*, related value L comes as $L = L^{Feed}{}_i = L^{Eat}{}_j$. Constraint (E5): $V^{Feed}{}_i = V^{Feed}{}_{i-1} + z_i.P_i - L^{Feed}{}_i$ augmented with constraint: $L^{Feed}{}_i \leq u_i.C^{Feed}$, is equivalent to (E2) and provides us with the evolution of quantities $V^{Feed}{}_i$. By the same way, constraint (E6): $V^{Eat}{}_j = V^{Eat}{}_{j-1} - x_j.S_j - v_j.S^{+}{}_j + L^{Eat}{}_j$, augmented with constraint $L^{Eat}{}_j \leq v_j.\ C^{Eat}$, is equivalent to (E3) and provides us with the evolution of quantities $V^{Eat}{}_j$. So we conclude for the first part of above statement.

As for the second part, about the optimal value of the rational relaxation of ***SFEK_MILP***, we get it by setting:

- For any $j = 1,\dots, M$: $v_j = ½$; $L^{Eat}{}_j = S_j + S^{+}{}_j/2$; $x_j = 1$; $V^{Eat}{}_j = H^{Eat}$;
- For any $i = 1,\dots, N$: $u_i = M/2N$; $z_i = 0$; $L^{Feed}{}_i = 0$; $V^{Feed}{}_i = H^{Feed}$;
- For any $i = 1,\dots, N$, $j = 1,\dots, M$: $U_{i,j} = 1/(2N)$.

If C^{+} is the null vector and if $S_j + S^{+}{}_j/2 \leq C^{Eat}.(2N - 1)/2N$ for any $j = 1,\dots, M$, then we get a feasible solution of our ILP model, which yields a null cost value. □

Remark 1: The conditions which underlie the second part of Theorem 1 are not very restrictive. In practice, the additional cost C^{+} will be most often significantly smaller than the main cost C. As for the inequality $S_j + S^{+}{}_j/2 \leq C^{Eat}.(2N - 1)/2N$, it is close to be a necessary condition for the existence of a feasible solution of **SFEK** model. The consequence of this gap induced by relaxing ***SFEK_MILP*** from its integrality constraints is that trying do efficiently deal with **SFEK** through the use of mathematical programming tools (CPLEX, …) will require an effort

in order to make appear well-fitted additional constraints or *cuts*. We are going to briefly discuss this issue in coming section III.1.

III.1. Enhancing SFEK_MILP.

In order to make above MILP formulation better fitted to numerical resolution, we enhance it by augmenting it with additional valid constraints. According to this purpose, we say that 2 pairs (i_1, j_1) an (i_2, j_2) are *antagonistic* iff $i_1 \leq i_2$ and $j_1 \geq j_2$, one at least of those 2 inequalities being strict. A collection Λ of pairwise *antagonistic* pairs (i, j) is called an *antagonistic clique*. Also, we introduce an auxiliary vector $F = (F_j \geq 0,\ j = 0,\ldots,\ M)$, with the intuitive meaning that F_j means the cumulative resource consumption by the *eater* from station 1 to station j. It comes that F follows the equations:

- $F_0 = 0$; For any $j \geq 1$, $F_j = F_{j-1} + x_j.S_j + v_j.S^+_j$.

We may now state the following lemmas 1, 2, 3, which provide us with respectively additional *feeder* constraints, *eater* constraints and *synchronization* constraints:

Lemma 1: *The following additional feeder constraints are valid*.

- $\Sigma_{1 \leq i \leq N} P_i.z_i \geq \Sigma_{1 \leq i \leq N} L^{Feed}_i \geq F_M$ (E10)
- $\Sigma_{1 \leq i \leq N} u_i \geq F_M/Inf(C^{Feed}, C^{Eat})$ (E11)

Proof: Clearly F_M represents a lower bound on the resource amount which must be globally transferred to *the eater.* Thus (E10) tells us that this resource must have been generated by the *feeder*.
As for (E11), we get it by noticing that the amount L_j of any *transfer* L^{Feed}_i related to a given period i cannot exceed neither C^{Feed} nor C^{Eat}. □

Lemma 2: *The following additional eater constraints are valid*.

- *For any* $j = 1,\ldots, M$: $F_j - H^{Eat} \leq \Sigma_{k \leq j} L^{Eat}_k$ (E12)
- $\Sigma_{1 \leq j \leq M} v_i \geq F_M/Inf(C^{Feed}, C^{Eat})$ (E13)

Proof: (E12) means that, for any j, resource $F_j - H^{Eat}$ must have been transferred by the *feeder* at stations 1,…, j. As for (E13), it comes exactly the same way as (E11). □

Lemma 3: *The following additional synchronization constraints are valid*:

- *For any antagonistic clique* Λ: $\Sigma_{(i,j) \in \Lambda} U_{i,j} \leq 1$ (E14)
- $\Sigma_{1 \leq i \leq N} L^{Feed}_i = \Sigma_{1 \leq j \leq M} L^{Eat}_j$ (E15)

(E14) can be separated in polynomial time.

Proof: The first part of this statement is trivial, since (E14) only extends constraints (E9) while (E15) expresses the fact the transfer are the same from both *eater* and *feeder* points of view. As for the separation issue, we notice that the *antagonistic* binary relation defines a partial order relation >>: (i_1, j_1) >> (i_2, j_2) iff $i_1 \leq i_2$ and $j_1 \geq j_2$, one at least of those 2 inequalities being strict, with (1, M) and (N, 1) as respectively unique maximal and minimal elements. It comes that, some vector U (rational or integral) being given, testing the existence of an *antagonistic* clique Λ such that $\Sigma_{(i,j) \in \Lambda} U_{i,j} > 1$ can be done by searching, through Bellman Algorithm, a chain with largest weight in the sense of the set $\Omega = \{1,\ldots, N\}.\{1,\ldots, M\}$, partially ordered by the relation >>, and whose elements (i, j) are provided with weights $U_{i,j}$. □

III.2. Computing a SFEK Lower Bound.

Let us first introduce the following **Aux_Eat** model:

Aux_Eat: {Compute {0, 1}-valued vectors $x = (x_j, j = 1,\ldots, M)$, $v = (v_j, j = 1,\ldots, M)$, together with non negative vectors $L^{Eat} = (L^{Lead}_j, j = 1,\ldots, M)$, $V^{Eat} = (V^{Eat}_j, j = 0,\ldots, M)$, in such a way that:

- $\Sigma_j L^{Eat}_j$ is minimal;
- For any j, $L^{Eat}_j \leq v_j.\mathrm{Inf}(C^{Eat}, C^{Feed})$;
- $\Sigma_j W_j.x_j \geq W^*$;
- $V^{Eat}_0 = 0$; For any $j \geq 1$, $V^{Eat}_j \leq C^{Eat}$.
- For any $j \geq 0$, $V^{Eat}_j = V^{Eat}_{j-1} - x_j.S_j + v_j.(L^{Eat}_j - S^+_j)$;
- $\Sigma_j L^{Eat}_j = \Sigma_j S_j.x_j + \Sigma_j S^+_j.v_j.\}$

The meaning of this model, which is a kind of augmented Knapsack model, is that the *eater player* behaves as if it were sure to receive resources whenever it needs them. So, it must only decide at which stations j it performs jobs and at which stations j and according to which amounts resource it asks for a transfer.

We may efficiently deal with **Aux_Eat** through dynamic programming, according to the search of a shortest path in the following auxiliary graph $G^{Eat} = (X^{Eat}, A^{Eat})$:

- A node x of X^{Eat} is a 3-uple ($j = 0,\ldots, M$, $V \geq 0$, $W \geq 0$), where $V \leq C^{Eat}$. If $j = M$, then we impose $W = 0$, $V = H^{Eat}$.

  Explanation: V has to be understood as the value V^{Eat}_j, and W as the value $\Sigma_{k \geq j} W_j.x_j$. This means that we are going to perform our path search while starting from $j = M$ and moving backward, while considering that we may do in such a way that when *eater* arrives in M, it only needs to be provided with the same resource amount H^{Eat} as when it started.
- Nodes (j-1, V, W) and (j, $V°$, $W°$) are connected by an arc a of A^{Eat}, corresponding to decisions x and v, if and only if one of the following configurations holds:
  - $W = W°$ and $V = V°$: in such a case $x = v = 0$ and related cost of a is null;
  - $W = \mathrm{Inf}(W^*, W° + W_j)$ and $V = V° + S_j$: in such a case $x = 1$, $v = 0$ and related cost of a is equal to S_j; For such an arc, we require $W° < W^*$;
  - $W = W°$ and $V = S^+_j$: in such a case $x = 0$, $v = 1$ and related cost of a is equal to S^+_j;
  - $W = \mathrm{Inf}(W^*, W° + W_j)$ and $V = S_j + S^+_j$: in such a case $x = 1$, $v = 1$ and related cost of a is equal to $S_j + S^+_j$; For such an arc, we require $W° < W^*$.

  Explanation: The 2 first cases correspond to the case $v = 0$, and only express the impact on V, W of a decision x = 1. The two last cases correspond to $v = 1$, and then we notice that we may do as if V were equal to 0 every time *eater* gets transfer value $L^{Eat}_j = V°$. This approach, which makes L^{Eat}_j not to be part of the decision, allows to significantly reduce the number of nodes of the graph G^{Eat} which are going to be generated during a backward driven Bellman path search process. This construction may be illustrated by the following figure 8:

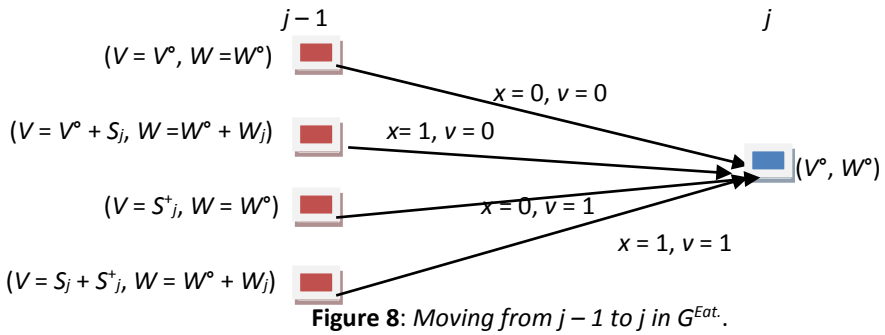

Figure 8: *Moving from j – 1 to j in* G^{Eat}.

Theorem 2: *Solving* ***Aux_Eat*** *means computing a path* Γ *in graph* G^{Eat} *which starts from some node (0,* $V \leq H^{Eat}$, *W*) and ends into node (M,* H^{Eat}, *0), and which is the shortest possible in the sense of above defined costs. It can be done in O(W*.* C^{Eat}.*M*).

Proof: The first part of our result is due to the fact that an optimal solution of **Aux_Eat** can always be computed in such a way that, for any j (but possibly the smallest one) such that $v_j = 1$, the value V^{Eat}_{j-1} is equal to $x_j.S_j + v_j.S^+_j$. The inequality $V \leq H^{Eat}$ required from starting nodes ($0, V \leq H^{Eat}, W^*$) ensures us that the sequence of decisions (x, v) related to optimal path Γ can be turned into an effective decision sequence for **Aux_Eat**.
In order to get the second part of our statement, we only need to notice that the number of nodes of the graph G^{Eat} does not exceed $W^*.C^{Eat}.M$ and that the outer degree of any such a node never exceeds 4. It comes that applying Bellman algorithm to this graph according to a backward strategy can be done in $O(W^*.C^{Eat}.M)$. □

Remark 2: In practice, the number of nodes generated throughout a backward driven implementation of Bellman shortest path search will involve far less nodes than $W^*.C^{Eat}.M$.

Then we see that we can compute a lower bound *LB_SFEK* for **SFEK** by dealing with **SFEK** while relaxing the synchronization constraints. This means that we first solve the **Aux-Eat** model and retrieve a cumulative transfer amount L^*, next computes vector z in order to minimize cost $C.z$ while achieving the *Feeder_Knapsack* constraint, and finally compute a lower bound $Q = \lceil L^*/\text{Inf}(C^{Eat}, C^{Feed}) \rceil$ for the smallest number of transfer transactions which are required to provide *eater* with L^* resources, together with their optimistic cost C^{+*} = Sum of the Q smallest C^+_i values. Related process may be summarized as follows:

LB_SFEK Procedure:
First Step: Solve **Aux_Eat** and denote by L^* its optimal value $\Sigma_j L^{Eat}_j$;
Second Step: Solve the following **Feeder_Knapsack** model:
Feeder_Knapsack: {Compute {0, 1}-valued vectors $z = (z_i, i = 1,\ldots, N)$ in such a way that:
- $\Sigma_i z_i.P_i \geq L^*$;
- $C.z = \Sigma_i z_i.C_i$ is minimal

Denote by C^* resulting optimal value;
Third Step: Set $Q = \lceil L^*/\text{Inf}(C^{Eat}, C^{Feed}) \rceil$ and C^{+*} = Sum of the Q smallest C^+_i values;
Then *LB_SFEK* comes as $C^{+*} + C^*$.

III.3. Speeding the Computation of a SFEK Lower Bound. .

Computing *LB_SFEK* lower bound according to III.2 may be time consuming and not fit the requirements of exact resolution algorithms. So we may skip **Aux_Eat** and proceed faster, in order to get a weaker lower bound *Weak_LB_SFEK*. More precisely, we are first going to get an estimation of global resource transfer L^* through the rational relaxation of the **Eater Knapsack** model : {Compute x such that $\Sigma_j W_j.x_j \geq W^*$ and which minimizes $\Sigma_j S_j$ }; Next we are going to get an estimation of the C^* value involved in the ***LB_SFEK*** procedure through the rational relaxation of the **Feeder_Knapsack** model involved into ***LB_SFEK***; Finally, we shall conclude by using L^* and C^* exactly the same way as in ***LB_SFEK***. Related process may be summarized as follows:

Weak_LB_SFEK Procedure:
First Step: Compute the optimal value of the rational relaxation of the following **Eater Knapsack** model:
Eater_Knapsack: {Compute {0, 1}-valued vectors $x = (x_j, j = 1,\ldots, M)$, such that:
- $\Sigma_j W_j.x_j \geq W^*$;
- $\Sigma_j S_j$ is minimal}

Denote by L^* resulting optimal value.
Second Step: Compute the optimal value of the rational relaxation of **Feeder Knapsack** model of III.2; Denote by C^* resulting optimal value.
Third Step: Proceed as in the third step of ***LB_SFEK***.

This process may be applied to the whole instance **SFEK**. But we shall check in next section IV that it may also be applied to any partial instance induced by some period i_0, some station j_0, and some current state of variables V^{Feed}, V^{Eat} and $W = \Sigma_{j \le j0} W_j.x_j$, in order to be embedded into a global dynamic programming algorithm.

IV. A PTAS (*Polynomial Time Approximation Scheme*) for SFEK.

We want to check here that, though **SFEK** is NP-Hard, it can be handled in an almost polynomial way. In order to do it, we first design a *Dynamic Programming* algorithm ***DP_SFEK***, which works in polynomial time once main parameters W^*, C^{Feed} and C^{Eat} are fixed. Then we get a polynomial time approximation scheme while filtering this algorithm by rounding devices. We proceed in 2 steps:

- In the first step, we describe algorithm ***DP_SFEK***.
- In the second step, we describe the way ***DP_SFEK*** may be adapted in order to work as a PTAS.

IV.1. The Algorithm DP_SFEK.

A dynamic programming algorithm usually relies on components which are its *time* space *T*, its *state* space Σ and a system of *decisions* and *transitions*. Transitions may be viewed as the arcs of an oriented graph whose node set is the Cartesian product $T.\Sigma$ and resulting DP algorithm becomes an implementation of Bellman shortest path algorithm for acyclic graphs. Let us first describe those components in the case of ***DP_SFEK***.

DP_SFEK Time Space *T*: It is the set of all pairs (i, j), $i = 0,\dots, N$, $j = 0, \dots, M$, that means the product of respectively the period set *I* and the station set *J*, both augmented with respectively a fictitious period 0 and a fictitious station 0. This *Time space T* is provided with its standard partial ordering σ: $(i, j)\ \sigma\ (i', j')$ iff $i + j < i' + j'$ or $(i + j = i' + j'$ and $i < i')$. We design ***DP_SFEK*** as a forward driven process, which scan *T* according to σ ordering.

DP_SFEK State Space Σ: For any time (i, j), a *state* s is a 4-uple $s = (W, V^{Feed}, V^{Eat}, Status)$ augmented with a cost value *Cost*. If we refer to the dynamics of **SFEK**, we see that the semantics of such a state *s* is that:

- *W* means the sum $\Sigma_{k \le j} W_k$;
- V^{Feed} and V^{Eat} mean V^{Feed}_i and V^{Eat}_j;
- *Cost* is the current cost $\Sigma_{k \le i} u_k.C^+_k. + \Sigma_{k \le i} z_k.C_k$;
- *Status* is a symbolic variable, which may take the following values:
  - *Status* = *E_Wait*: It means that *eater* is waiting for *feeder* to transfer resources and that it has previously decided about x_j and v_j : at this time, value V^{Eat} may be negative, since transfer amount L^{Eat}_j remains to be taken into account;
  - *Status* = *F_Wait*: It means that *feeder* is waiting for *eater* to ask for resources;
  - *Status* = *E_End*: It means that *eater* achieved value W^* and that current value V^{Eat} is at least equal to H^{Eat}. Thus no decision (x, v) is going to be taken any more. Still, we have that $V^{Feed} < H^{Feed}$ and so that the process must go on in order to make feeder achieve the inequality $V^{Feed} \ge H^{Feed}$;
  - *Status* = *End*: We have $V^{Feed} \ge H^{Feed}$ and $V^{Eat} \ge H^{Eat}$ and $W \ge W^*$.

According to this we impose $0 \le W$, $0 \le V^{Feed} \le C^{Feed}$ and $-\text{Inf}(C^{Feed}, C^{Eat}) \le V^{Eat} \le C^{Eat}$. We also impose that if $i = N$ then *Status* = *End*. If it is not the case, then we are in a *Fail* configuration and related state may be *killed* (removed). Initial state corresponds to $i = j = 0$ and $s = (0, H^{Feed}, H^{Eat}, F\_Wait)$ with cost value *Cost* = 0. Final states are states $s = (W, V^{Feed}, V^{Eat}, End)$ such that $V^{Feed} \ge H^{Feed}$ and $V^{Eat} \ge H^{Eat}$ and $W \ge W^*$.

Decisions: Then a decision *d* applied to a state $s = (W, V^{Feed}, V^{Eat}, Status)$ at time (i, j) is a 4-uple $d = (z, u, x, v)$ in $\{0, 1\}^4$, with the meaning:

- $z = 1$ means $z_{i+1} = 1$; $u = 1$ means $u_{i+1} = 1$;

- $x = 1$ means $x_{i+1} = 1$; $v = 1$ means $v_{i+1} = 1$.

Feasible Decisions and Related Transitions: Several cases must be considered:

1) *Status = End*: Then we are at the end of the process and no decision can be taken.
2) *Status = E_End*. Then x, u and v are neutralized, and the only decision is about z: $z = 1$ requires $V^{Feed} + P_{i+1} \leq C^{Feed}$. In any case, we shift to $(i+1, j)$, with resulting state $(W, V^{Feed} + z.P_{i+1}, V^{Eat,}, Status1)$ and transition cost $z.C_{i+1}$, with *Status1* given by: If $V^{Feed} + z.P_{i+1} \geq C^{Feed}$ then *Status1* = *End* else *Status1* = *Status*;
3) *Status = E_Wait*. Then decision is about z and u, while x and v are neutralized:
   - $(z \in \{0, 1\}, u = 0)$ requires $V^{Feed} + z.P_{i+1} \leq C^{Feed}$ and $i \leq N - 1$ Then we shift to $(i+1, j)$, with resulting state $(W, V^{Feed} + z.P_{i+1}, V^{Eat,}, Status)$ and transition cost $z.C_{i+1}$;

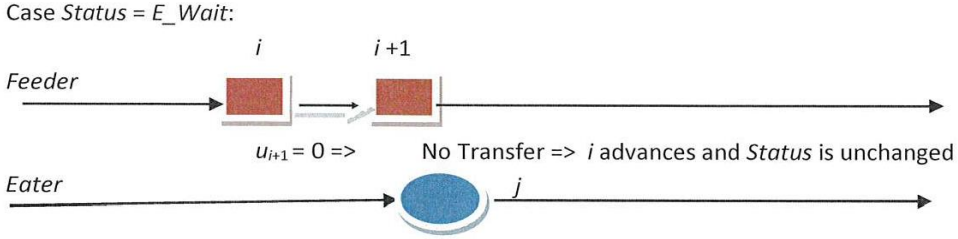

Figure 4: *Status = E_Wait and u = 0*.

   - $(z \in \{0, 1\}, u = 1)$ means that a resource amount equal to $L = \text{Inf}(C^{Feed}, C^{Eat}, C^{Eat} - V^{Eat}, V^{Feed} + z.P_{i+1})$ is going to be transferred. It requires $V^{Feed} + z.P_{i+1} - L \leq C^{Feed}$ and $L + C^{Eat} \geq 0$, together with $i \leq N - 1$. Transition cost is equal to $z.C_{i+1} + C^{+}_{i+1}$. Then we shift to $(i+1, j)$, with resulting state $(W, V^{Feed} + z.P_{i+1} - L, L + V^{Eat,}, Status1)$, with *Status1* given by:
     - If $(W \geq W^*) \wedge (L + V^{Eat,} \geq H^{Eat}) \wedge (V^{Feed} \geq H^{Feed})$ then *Status1* = *End*;
     - If $(W \geq W^*) \wedge (L + V^{Eat,} \geq H^{Eat}) \wedge (V^{Feed} < H^{Feed})$ then *Status1* = *E_End*;
     - Else *Status1* = *F_Wait*.

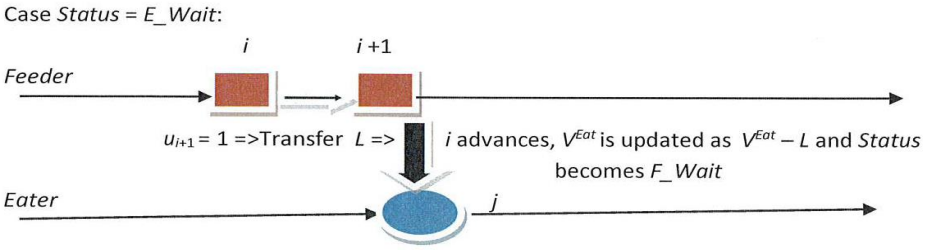

Figure 5: *Status = E_Wait and v = 1*.

4) *Status = F_Wait*. Then decision is about x and v, while z and u are neutralized:
   - $(x \in \{0, 1\}, v = 0)$ requires $V^{Eat} - x.S_{i+1} \geq 0$. Then we shift to $(i, j+1)$, with resulting state $(W + x.W_{j+1}, V^{Feed}, V^{Eat,} - x.S_{i+1}, Status1)$ and null transition cost. *Status1* is given by:
     - If $(W + x.W_{j+1} \geq W^*) \wedge (V^{Eat,} - x.S_{i+1}, \geq H^{Eat}) \wedge (V^{Feed} \geq H^{Feed})$ then *Status1* = *End*;
     - If $(W + x.W_{j+1} \geq W^*) \wedge (V^{Eat,} - x.S_{i+1}, \geq H^{Eat}) \wedge (V^{Feed} < H^{Feed})$ then *Status1* = *E_End*;
     - Else *Status1* = *Status*;

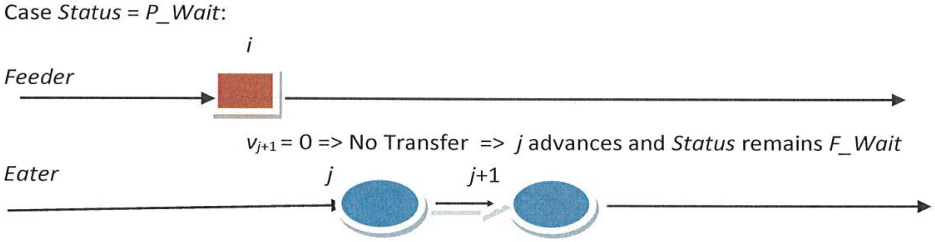

Figure 6: *Status = F_Wait and v = 0.*

- ($x \in \{0, 1\}$, $v = 1$) means that a resource transfer is going to take place at $j + 1$, according to a resource amount L which remains undetermined. It requires $V^{Eat} - x.S_{j+1} - S^{+}_{j+1} \geq -\text{Inf}(C^{Feed}, C^{Eat})$, together with $i \leq N - 1$. Then we shift to $(i, j+1)$, with resulting state $(W + x.W_{j+1}, V^{Feed}, V^{Eat,} - x.S_{j+1} - S^{+}_{j+1}, E\_Wait)$ and null transition cost.

  Remark 3: In this last case *Status* = *F_Wait*, $v = 1$, we see that resulting value V^{Eat} may be negative. The fact is that at the time when related transition is performed, final value V^{Eat} is not completely determined, since transfer amount L is still not know. V^{Eat} will come back to be non negative as soon as L will be computed according to the case *Status* = *E_Wait*, $u = 1$.

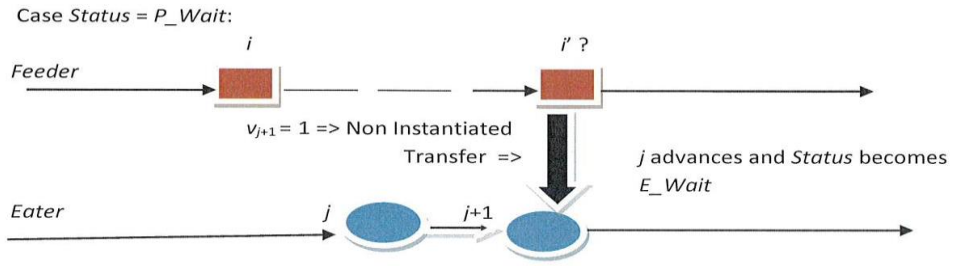

Figure 7: *Status = F_Wait and v = 1.*

For any time (i, j) and any state *s,* we denote by $DEC((i, j), s)$ the set of all decisions which are feasible for (i, j) and *s*, and so which yield a feasible resulting state. Of course, if *s* is not a final state and $DEC((i, j), s)$ is empty, then we are in a *Fail* configuration for *state s*, which we may *kill*. Then a forward driven implementation of Bellman equations to above framework comes as follows

***DP_SFEK* Algorithm**:

- Initialize the sets $State(i, j)$ of states related to time pair (i, j) as:
  - $State(0, 0) = \{s_0 = (0, H^{Feed}, H^{Eat}, P\_Wait)$, augmented with cost value *Cost* = 0};
  - For any $(i, j) \neq (0, 0)$, set $State(i, j)$ = Nil = Empty Set;
- Scan the time space *T* according to linear order σ, and, for every time value (i, j), consider all states $s = (W, V^{Feed}, V^{Eat}, Status)$ in $State(i, j)$, together with their cost value *Cost*, and all decisions *d* in $DEC((i, j), s)$ and compute resulting time value (i', j'), resulting state $s' = (W', V^{Feed'}, V^{Eat'}, Status')$ together with related cost value *Cost'* = *Cost* + (cost of the transition $((i , j), s) \rightarrow ((i' , j'), s')$). Then:

  In case no $s_1 = (W_1, V^{Feed}{}_1, V^{Eat}{}_1, Status_1) = s' = (W', V^{Feed'}, V^{Eat'}, Status')$ then insert *s'* into $S(i', j')$

  Else, s_1 being as above and $Cost_1$ being related cost value

  If $Cost_1 <$ *Cost'* then replace s_1 by *s'*. (I1)
- Retrieve (i, j) and $s = (W, V^{Foll}, V^{Lead}, Status)$ in $State(i, j)$ such that:
  - *Status* = *End*;
  - Related value *Cost* is minimal.

  If (i, j) and *s* do not exist then *DP_SFEK* = *Fail* else retrieve (z, u, x, k), together with a transfer amount vector *L*, related to (i, j) and *s*, and set *DP_SFEK* = (z, u, x, k, L).

Theorem 3: ***DP_SFEK*** *solves* ***SFEK*** *in pseudo-polynomial time, that means in polynomial time once we decide to bound coefficients* C^{Feed}, C^{Eat} *and* W^*.

Proof: The fact that ***DP_SFEK*** solves **SFEK** in an exact way is a matter of routine. It mainly derives from the fact that an optimal solution $(z, x, Trans)$ may always be computed in such a way that, for any transfer transaction

(i, j, L) in *Trans*, value L is maximal and equal to $L = \text{Inf}(C^{Feed}, C^{Eat}, C^{Eat} - V^{Eat}, V^{Feed} + z.P_{i+1})$ as in above case *Status* = *E_Wait*, $u = 1$.
In order to get the complexity part, we first notice that we may replace any W_j such that $W_j \geq W^*$ by W^*. Then we see that states (W, V^{Feed}, V^{Eat}, *Status*) which we handle are such that $0 \leq V^{Feed} \leq C^{Feed}$, $-C^{Eat} \leq V^{Eat} \leq C^{Eat}$ and $0 \leq W \leq W^*$. It comes that for any (i, j), the number of related states cannot exceed $8.C^{Feed}.C^{Eat}.W^*$. Besides, the number of decisions which may be tried from a given state never exceeds 4. It comes that if we implement Bellman principle while storing sets *State*(i, j) into tables, then we get a process which runs in no more than $O(M.N.\ C^{Feed}.C^{Eat}.W^*)$. □

IV.2. Enhancing DP_SFEK with Pruning rules.

In order to make decrease the number of states which are going to really be used throughout the execution of ***DP_SFEK***, we may implement the following pruning rules:

- <u>*Domination* Rule</u>: If 2 states $s = (W, V^{Feed}, V^{Eat}, Status)$ and $s' = (W', V^{Feed'}, V^{Eat'}, Status')$ in *State*(i, j), respectively provided with cost values *Cost* and *Cost*', are such that: $W \leq W'$; $V^{Eat'} \leq V^{Eat}$; *Status* = *Status'*, then we may *kill* (remove) s' from *State*(i, j).
  Remark 4: Above notion of domination does not involve V^{Feed}, since the fact for V^{Feed} to be large may forbid decision $z_i = 1$, at a time when such a decision would be the best one.

- <u>*Logical Eater* Rule</u>: If some states $s = (W, V^{Feed}, V^{Eat}, Status)$ in *State*(i, j) is such that $W + \sum_{k \geq j+1} W_k < W^*$, then we may kill s, since it will forbid *eater* from achieving the constraint $\sum_j x_j.W_j \geq W^*$.

- <u>*Logical Feeder* Rule</u>: For any state $s = (W, V^{Feed}, V^{Eat}, Status)$ in *State*(i, j) let us set:
  - L^* = the value of the rational relaxation of the ***Aux_Eater_Knapsack*** program:
    - $\sum_{k \geq j+1} x_k.W_k \geq W^* - W$;
    - Minimize $\sum_{k \geq j+1} x_j.S_k$;
  - $Q = \lceil (L^* - V^{Eat} + H^{Eat})/\text{Inf}(C^{Feed}, C^{Eat}) \rceil$ and L^{*+} = the sum of the Q smallest S^+_k, $k \geq j + 1$, coefficients.

  If $\sum_{k \geq i+1} P_k < L^* + L^{*+} + H^{Feed} - C^{Feed}$ then we may *kill* s, since it will forbid *Feeder* from achieving the constraints $\sum_i z_i.P_i \geq \sum_j x_j.S_j + \sum_j x_j.S^+_j$, $V^{Feed}_N \geq H^{Feed}$.

- <u>*Lower/Upper Bound* Rule</u>: Let us suppose that we are provided with a feasible solution (z, x, *Trans*) of **SFEK**, with cost value *Cost**. We may have obtained such a solution through any kind of greedy or local search heuristic. Then, L^*, L^{*+} and Q being defined as in the *Logical Feeder* rule, let us denote by C^{+*} the sum of the Q smallest coefficients C^+_k, $k \geq i + 1$ and by C^* the optimal value of the rational relaxation of the following **Aux_Feeder_Knapsack** program:
  - $\sum_{k \geq i+1} z_k.P_k \geq L^* + L^{*+} + H^{Feed} - C^{Feed}$;
  - Minimize $\sum_{k \geq i+1} z_k.P_k$.

  If $C^* + C^{+*} \geq Cost^*$, then we *kill* s, since it forbids us to improve current feasible solution (z, x, *Trans*).

IV.3. Deriving a PTAS from DP_SFEK.

In order to derive such an approximation scheme, we first need to introduce some definitions:

- We call *full state* any 5-uple $sf = (W, V^{Feed}, V^{Eat}, Status, Cost)$ which combines a state $s = (W, V^{Feed}, V^{Eat}, Status)$ and a cost value *Cost*.
- Let K be some positive integral number. If $h = a_0 + a_1.2 + \ldots + a_q.2^q$ is some non negative integral number, then we set: If $q \leq K$ then $Round(h, K) = h$ else $Round(h, K) = a_{q-K}.2^{q-K} + \ldots + a_q.2^q$. If h is a negative integral number, then we set $Round(h, K) = -Round(h, K)$.

- If 2 numbers h_1 and h_2 are such that $Round(h_1, K) = Round(h_2, K)$, then we say that they are *equivalent modulo K*.
- By extension, 2 *full states* $sf_1 = (W_1, V^{Feed}{}_1, V^{Eat}{}_1, Status_1, Cost_1)$ and $sf_2 = (W_2, V^{Feed}{}_2, V^{Eat}{}_2, Status_2, Cost_2)$ are equivalent *modulo K* if $Status_1 = Status_2$, $Round(W_1, K) = Round(W_2, K)$, $Round(V^{Feed}{}_1, K) = Round(V^{Feed}{}_2, K)$, $Round(V^{Eat}{}_1, K) = Round(V^{Eat}{}_2, K)$ and $Round(Cost_1, K) = Round(Cost_2, K)$. In such a case we write: $sf_1 = sf_2$ *Modulo K*.

In order to turn ***DP_SFEK*** into an approximation scheme ***DP_SFEK_PTAS*** depending on an integral approximation parameter *H*, we must take into account the fact that, since values V^{Feed} and V^{Eat} may become arbitrarily small throughout the ***DP_SFEK*** process, rounding transfer value *L* may turn them into negative values. It comes that we must accept an error margin not only on the final cost value, but also on the way capacity and initial/final constraints are satisfied. As a matter of fact, ε being such an error margin, ***DP_SFEK_PTAS***(*H*) applied with $2^{-H} \leq \varepsilon$ will compute in polynomial time a sequence of decision Π which achieves target utility *W**, under a global cost no larger than $Cost^{Opt}.(1+\varepsilon)$, and in a way which is consistent with final constraints and with relaxed capacity and initial constraints. While relaxing capacity constraints will mean replacing C^{Feed} and C^{Eat} respectively by $C^{Feed}.(1 + \varepsilon)$ and $C^{Eat}.(1 + \varepsilon)$, which will mean a relative error, relaxing initial constraints will mean increasing initial V^{Feed} and V^{Eat} values H^{Feed} and H^{Eat} respectively by $C^{Feed}.\varepsilon$ and $C^{Eat}.\varepsilon$, which will correspond to an absolute error.

More precisely, approximation scheme ***DP_SFEK_PTAS***(*H*) is going to be an adaptation of ***DP_SFEK*** which works on an **SFEK** instance updated as follows:

- Capacity constraints are related to $C^{Feed}.(1 + 2^{-H})$ and $C^{Eat}.(1 + 2^{-H})$;
- Initial constraints about V^{Feed} and V^{Eat} are any integral number between respectively $\lfloor H^{Feed} + C^{Feed}.2^{-H}/4 \rfloor$ and $\lfloor H^{Feed} + 3.C^{Feed}.2^{-H}/4 \rfloor$, and $\lfloor H^{Eat} + C^{Eat}.2^{-H}/4 \rfloor \lfloor H^{Eat} + 3.C^{Eat}.2^{-H}/4 \rfloor$; Final constraints remain the same;
- Constraint about target utility *W** remains the same.

In order to derive ***DP_SFEK_PTAS***(*H*) from ***DP_SFEK***, we apply the following modifications:

- <u>Modifications of the components of the algorithm</u>:
  - We extend the notion of *decision*: every time a t*ransfer* decision $u = 1$ is taken, related transfer amount *L* becomes part of the decision, with encoding size limited to *K*+1 bits and value no larger than $Inf(C^{Feed}.(1 + 2^{-H}), C^{Eat}.(1 + 2^{-H}))$. So an *extended decision* means a 5-uple (*z*, *u*, *x*, *v*, *L*);
  - State sets *S*(*i*, *j*) becomes *full state* sets *SF*(*i*, *j*).

- <u>Modifications of the structure of the algorithm</u>.
  - We first derive from *H* a rounding value *K*: More precisely we set $n = N+M+1 = 2^R + a_{H-1}.2^{R-1} + \ldots + a_0$. and $K = (H + R + 5)$. (I2)
  - Next, in the Bellman ***DP_SFEK*** process, we replace instruction (I1) by following instruction (I1*) which forbids, for any (*i*, *j*), 2 *full states* in set *SF*(*i*, *j*) to be equivalent modulo *K* :

    <u>Instruction (i1*):</u>
    For every $sf = (W, V^{Feed}, V^{Eat}, Status, Cost)$ in *full state* set *SF*(*i*, *j*), and every decision *D* = (*z*, *u*, *x*, *v*, *L*) in *DEC*((*i*, *j*), *sf*) do
    Compute resulting time (*i'*, *j'*) and resulting *full state* $sf' = (W', V^{Feed\prime}, V^{Eat\prime}, Status', Cost')$.
    If no $sf_1 = (W_1, V^{Feed}{}_1, V^{Eat}{}_1, Status_1, Cost_1)$ exists in *SF*(*i'*, *j'*), such that $(W_1, V^{Feed}{}_1, V^{Eat}{}_1, Status_1, Cost_1) = (W', V^{Feed\prime}, V^{Eat\prime}, Status', Cost')$ *Modulo K* then Insert *sf'* into *SF*(*i'*, *j'*)
    Else, sf_1 being as above,
    If $W_1 < W'$ then Replace sf_1 by *sf'*. (I1*)

It comes now that we may state:

SFEK PTAS Theorem 4: *For any **SFEK** instance, **DP_SFEK_PTAS**(H) computes in $O(N.M.2^{5H}.(N + M)^5)$ a sequence Π of extended decisions (z, u, x, v, L) together with a value $Cost(\Pi)$, which satisfies:* (E16)

- *Π is consistent with capacities by $C^{Feed}.(1 + 2^{-H})$ and $C^{Eat}.(1 + 2^{-H})$;*
- *Π is consistent with initial resource integral values HP^{Feed}, HP^{Eat} respectively between respectively $\lfloor H^{Feed} + C^{Feed}.2^{-H}/4 \rfloor$ and $\lfloor H^{Feed} + 3.C^{Feed}.2^{-H}/4 \rfloor$, and $\lfloor H^{Eat} + C^{Eat}.2^{-H}/4 \rfloor$ and $\lfloor H^{Eat} + 3.C^{Eat}.2^{-H}/4 \rfloor$, and with final resource values H^{Feed} and H^{Eat};*
- *Π is consistent with target utility W*;*
- *$Cost(\Pi) \leq (1 + 2^{-H}).Cost^{Opt}$, where $Cost^{Opt}$ denotes the optimal value of the instance.*

Proof: In order to prove that extended decision sequence Π satisfies (E16), we consider some optimal extended decision sequence Π^{Opt}, as it may derive from an application of ***DP_SFEK*** to our **SFEK** instance. Π^{Opt} is related to some *time* sequence $\{(i_0 = 0; j_0 = 0), (i_1, j_1), \ldots, (i_Q, j_Q)$ and some *full state* sequence $\{sf^{Opt}{}_0, sf^{Opt}{}_1, \ldots, sf^{Opt}{}_q = (W^{Opt}{}_q, V^{Feedl\_Opt}{}_q, V^{Eat\_\ Opt}{}_q, Status^{Opt}{}_q, Cost^{Opt}{}_q), \ldots, s^{Opt}{}_Q\}$. Then we proceed by induction, and check that, for any q, there exists a *full state* $sf = (W, V^{Feed}, V^{Eat}, Status, Cost)$ generated in $SF(fi_q, j_q)$ by ***DP_SFEK_PTAS***(H), such that: (E17)

- $W \geq W^{Opt}{}_q$;
- $Cost \leq (1 + q.2^{-K}).Cost^{Opt}{}_q$;
- $\| V^{Feed\_Opt}{}_q - V^{Feed} \| - | H^{Feed} - HP^{Feed} \| \leq q.C^{Feed}.2^{H-R-4}$;
- $\| V^{Eat\_Opt}{}_q - V^{Eat} \| - | H^{Eat} - HP^{Eat} \| \leq q.C^{Eat}.2^{H-R-4}$.

If we can do it, we clearly get the (E16) part of Theorem 4, which coincides with (E17) equations with $q = n$.
Notice that (E17) ensures V^{Feed} as well as V^{Eat} to be non negative and consistent with the capacity constraints as relaxed in (E16). Initial values $V^{Feed} = H^{Feed} + C^{Feed}.2^{-H}$ and $V^{Eat} = H^{Eat} + C^{Eat}.2^{-H}$ make the case $q = 0$ trivial. So let us suppose that this true for some q. Then we consider extended decision $d_q{}^{Opt} = (z_q{}^{Opt}, u_q{}^{Opt}, x_q{}^{Opt}, v_q{}^{Opt}, L_q{}^{Opt})$ which is associated wih sf_q. If $u_q{}^{Opt} = 0$ then $L_q{}^{Opt} = 0$ else we have $L_q{}^{Opt} = \text{Inf}(C^{Feed}, C^{Eat}, C^{Eat} - V^{Eat\_\ Opt}{}_q, V^{Feed\_Opt}{}_q + z_q{}^{Opt}..P_{i+1})$.
What we do now is to make ***DP_SFEK_PTAS***(H) apply to current state $sf = (W, V^{Feed}, V^{Eat}, Status, Cost)$ in $SF(fi_q, j_q)$ an extended decision $d = (z, u, x, v, L)$ which emulates $d_q{}^{Opt} = (z_q{}^{Opt}, u_q{}^{Opt}, x_q{}^{Opt}, v_q{}^{Opt}, L_q{}^{Opt})$. So, starting from full state *sf* as i (E17), we proceed as follows:

- If $u_q{}^{Opt} = 0$ then we apply decision $d = d_q{}^{Opt}$ to *sf* and get a full state sf^{New} to be possibly inserted into $SF(i_{q+1}, j_{q+1})$.
- If $u_q{}^{Opt} = 1$ then we round $L_q{}^{Opt}$ modulo *K*, get a rounded value $L^*{}_q{}^{Opt}$, apply the decision $d = (z_q{}^{Opt}, u_q{}^{Opt}, x_q{}^{Opt}, v_q{}^{Opt}, L^*{}_q{}^{Opt})$ to *sf* and get a full state sf^{New} to be possibly inserted into $SF(i_{q+1}, j_{q+1})$.

In both case, (E17) makes that decision *d* is feasible. Then the insertion procedure (I1*) is applied to state $sf^{New} = (W^{New}, V^{Feed\_New}, V^{Eat\_New}, Status^{New}, Cost^{New})$ which means that, at the end of the process, we get that either sf^{New} belongs to $SF(i_{q+1}, j_{q+1})$ or that there exists $sf°$ in $SF(i_{q+1}, j_{q+1})$ such that $(W°, V^{Feed}°, V^{Eat}°, Status°, Cost°)$ is equivalent to sf^{New} modulo K. In such a case, (I1*) implies makes that that $W° \geq W^{New}$.
Then, in order to achieve the proof of (E16), we must check that (E17) holds for sf^{New} in case sf^{New} in inserted into $SF(i_{q+1}, j_{q+1})$ and for $sf°$ in case $sf° = (W°, V^{Feed}°, V^{Eat}°, Status°, Cost°)$ equivalent to sf^{New} modulo *K* and such that $W° \geq W^{New}$, exists in $SF(i_{q+1}, j_{q+1})$. We do it by checking all inequalities involved into (E17):

- Inequality related to *Cost*: Since *Cost* values evolve in a monotonic way throughout any sequence of extended decisions, this case works as standard PTAS. In order to get sf^{New} while shifting from (i_q, j_q) to (i_{q+1}, j_{q+1}), we add the value $z_q{}^{Opt}.C_{i(q+1)} + z_q{}^{Opt}.C^+{}_{i(q+1)}$ to *Cost*., and this does not make increase the relative error, which does not exceed $q.2^{-K}$ according to (E17). Possibly replacing resulting $Cost^{New}$ by $Cost°$ equivalent to $Cost^{New}$ modulo *K*, cannot make resulting relative error exceed $(q+1).2^{-K}$.
- Inequality related to *W*: It works as for *Cost*, since *W* values also evolve in a monotonic way throughout any sequence of extended decisions. Induction hypothesis related to (E17) tells us that *W*

$\leq W^{Opt}_q$, and so we deduce $W^{New} \leq W^{Opt}_{q+1}$. The way (I1*) is implemented yields that, if we replace sf^{New} by sf° as mentioned above, then we shall keep $W^{\circ} \leq W^{New} \leq W^{Opt}_{q+1}$.

- Inequality related to V^{Eat}: Since V^{Eat} evolves in a non monotonic way and may take small values, we must deal here with absolute errors. Let us set $C^{Eat} = 2^B + \ldots + b_0 \leq 2^{B+1}$. Then $L_q^{Opt} = \text{Inf}(C^{Feed}, C^{Eat}, C^{Eat} - V^{Eat\_Opt}_q, V^{Feed\_Opt}_q + z_q^{Opt}..P_{i+1})$ is bounded by 2^{B+1}. Also, we have that $(1 + 2^{-H}).C^{Eat} \leq 2^{B+2}$. It comes that, at any time during the process, current value V^{Eat} will not exceed 2^{B+2}. Rounding such a value any modulo K means a relative error no more that $2^{-(K+1)}$ and an absolute error which does not exceed $2^{B+2}.2^{-(K+1)} = 2^{B-H-R-4}$. That means that, at every step during the process, we have that:
  - If $u_q^{Opt} = 0$, then either V^{Eat} and V^{Opt}_q remains both as they are when we shift from (i_q, j_q) to (i_{q+1}, j_{q+1}) or they both decrease by $S_{j(q+1)}$. In such a case, absolute error between V^{Eat_New} and V^{Opt}_{q+1} remains unchanged. If the rounding process makes us replace V^{Eat_New} by $V^{Eat\circ}$, then related absolute error increases by no more than $2^{B-H-R-4}$.
  - If $v_q^{Opt} = 1$, then V^{Eat} and V^{Opt} augment respectively by $L^*_q{}^{Opt}$ and L_q^{Opt}. By the same way, the absolute error between $L^*_q{}^{Opt}$ and L_q^{Opt} does not exceed $2^{B-H-R-4}$ and so absolute error between V^{Eat_New} and V^{Opt}_{q+1} does not increase by more than with respect to the absolute error between V^{Eat} and V^{Opt}_q. If the rounding process makes us replace V^{Eat_New} by $V^{Eat\circ}$, then related absolute error increases by no more than $2^{B-H-R-3}$.

  Repeating this no more than $N + M + 1 \leq 2^{R+1}$ times induces an absolute error which cannot exceed $2^{R+1}.2^{B-H-R-3} = 2^{B-H--2}$, that means which cannot exceed $2^{-H}.C^{Eat}/4$. Then we deduce (E17).
- Inequality related to V^{Feed}: It works as for V^{Eat}.

What remains to be proven is that, H being given, ***SFEK_PTAS***(H) works in $O(2^{5H}.(N + M)^6)$ on our ***SFEK*** instance. In order to get it, we first notice that, for any (i, j), the number of *full states* $sf = (W, V^{Feed}, V^{Eat}, Status, Cost)$ in $SF(i, j)$ is bounded, by $2.4.2^{4K+4} = 8.2^{4K+4}$. By the same way, the number of possible decisions (z, u, x, v, L) is bounded by $4.2.\ 2^{K+1} = 8.2^{K+1}$. Since $2^{K+1} = 2^K.2^{R+4} = 16.(N+M+1)^4.2^H$, we get that, for a given (i, j), we do not try more than $O(2^{5H}.(N + M)^5)$ transitions. Trying and applying a transition can be performed in constant time. So we conclude, while using the fact that the number of time values (i, j) that we have to handle during the process does not exceed $N.M + 1$. □

V. Conclusion: A Brief Discussion about Collaborative Approaches.

We just presented here models, algorithms and approximation results designed according to what we may call the *centralization* paradigm. Still, if we refer to practice, and the kind of situations which may involve variants of **SFEKP**, we guess that those situations may induce the need for a kind of cooperation between both *eater* and *feeder players*.

An approach for the adaptation of our algorithms to such a *collaborative* context may consist in first solving the ***Aux_Eat*** model presented in Section III.2 and next turn resulting solution into a package of constraints to be transmitted to the *feeder* player. Then this *feeder* player would compute its own strategy (z, u), and possibly send back its own constraints and criteria (*counter proposal*) to the *eater*.

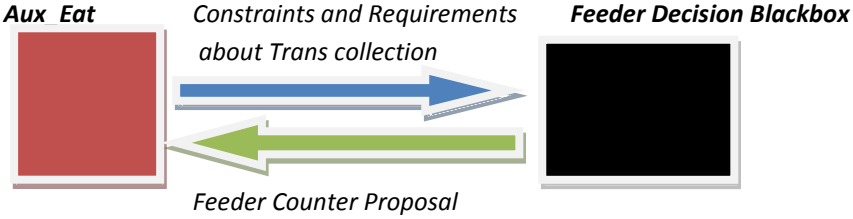

Figure 9: *A Feeder/Eater Collaborative Scheme*.

Above decomposition scheme may be implemented as a one-way scheme (*pipe-line* => the *eater* player acts as a leader), as a hierarchical scheme (one among both players takes the role of the leader and acts as in Bender's decomposition scheme), or an *horizontal* scheme (nobody is a natural leader). In such a case a pricing mechanism might be introduced in order to help into making the process converge: This pricing mechanism would search for a way to turn *eater* target W_j values into *feeder* cost values and so make the global cost Σ_i ($z_i.C_i + u_i.C^+_i$) to be fairly distributed between both players.

In case **SFEK** were extended to several *feeder* or *eater* players, more sophisticated approaches should be tried, involving for instance the use of the *Game Theory* framework. Our purpose for the future is to go further with the handling of such interactions and also to deal with the **SFEK** extensions which involve time.

VI. Bibliography.

[1]. S.ALBERS: *On energy conservation in data centers*; In **Proc. SPAA 2017, Washington D.C**, p 35-44, (2017).

[2]. P.AVELLA, M.BOCIA, I.VASILYEV : *A branch and cut algorithm for the multi-level generalized assignment problem* ; **IEEE Access** 1, p 475-479, (2013).

[3]. E.BAMPIS, B.ESCOFFIER, A.TEILLER: *Multistage Knapsack*; **arXiv**:1901.11260v1, 18 pages, (2019).

[4]. L.BENINI, A.BOGLOLO, G.DE MICHELI: *A survey of design techniques for system level dynamic power management*; **IEEE Transactions of Very Large Scale Integration Systems**, 8, 3, p 299-316, (2000).

[5]. K.BIEL, G.H.GLOCK : *Systematic literature review of decision support models for energy efficient decision planning*; **Computer and Industrial Engineering** 101, p 243-259, (2016).

[6]. L.BROTCORNE, S.HANAFI, R.MANSI: *A dynamic programming algorithm for the bi-level Knapsack problem*; **Operations Research Letters** 37, p 215-218, (2009).

[7]. A.CAPRARA, M.CARVALHO, A.LODI, G.J.WOEGINGER: *A study on the computational complexity of the bilevel knapsack problem*; **SIAM Journal on Optimization** 24 (2), p 823-838, (2014).

[8]. A.CAPRARA, H.KELLERER, U.PFERSCHY, D.PISINGER: *Approximation algorithms for Knapsack problems with cardinality constraints*; **EJOR** 123 (2), p 333-345, (2000).

[9]. L.CHEN, G.ZHANG: *Approximation algorithms for a bi-level Knapsack problem*; **Theoretical Computer Sciences** 497, p 1-12, (2013).

[10]. A.CLARK, B.ALMADA LOBO, C.ALMEDER: *Lot Sizing and Scheduling: Industrial extensions and research opportunities*; **International Journal of Production Research** 49 (9), p 2457-2461, (2011).

[11]. B.COLSON, P.MARCOTTE, G.SAVARD: *Bi-level programming: A survey*; **40R** 3, p 87-107, (2005).

[12]. S.DEMPE, K.RICHTER: *Bi-level programming with Knapsack constraints*; **CEJOR** 8, p 93-107, (2000).

[13]. A.DREXL, A.KIMMS: *Lot Sizing and Scheduling: Survey and extensions*; **EJOR** 99 (2), p 221-235, (1997).

[14]. T.DUDAS, B.KLINZ, G.J.WOEGINGER: *The computational complexity of multi-level bottleneck programming problems*; **In Multilevel Optimization: Algorithms and Applications; A.MIGDALAS, P.M.PARDALOS, P.VARBRAND Eds, Kluwer Acad. Publishers**, p 165-179, (1998).

[15]. A.FREVILLE: *The multi-dimensionnal 0-1 Knapsack problem: An Overview*; **EJOR** 155, p 1-21, (2004).

[16]. G.GOISQUE, C.RAPINE: *An efficient algorithm for the 2-level capacitated lot-sizing problem with identical capacities at both levels*; **EJOR** 261, p 918-928, (2017).

[17]. C.GOLDMAN: *Coordination of energy efficiency and demand response*; **Lawrence BERKELEY National Lab. Research Report**, Berkeley, CA, (2010).

[18]. S.HELBER, F.SAHLING: *A fix and optimize approach for the multi-level capacitated lot sizing problem*; **ECONSTOR, ZBW Diskussion Beitrag** 293, **Leibnitz Universitat Hannover**, 22 pages, (2008).

[19]. Y-F.HUNG, K-L.CHIEN: *A multi-class multi-level capacitated lot sizing model*; **Journal of the Operational Research Society**, 51-11, p 1309-1318, (2000).

[20]. P.KAMONSKY, D.SIMCHI-LEVI: *Production and distribution Lot Sizing in a two-stage supply chain*; **IIE Transactions** 35 (11), p 1065-1075, (2003).

[21]. H.KELLERER, U.PFERSCHY, D.PISINGER: *Knapsack problems*, **Springer-Verlag Berlin**, Heidelberg (2004).

[22]. M.HIFI: *Approximate algorithms for the container problem*; **Operations Research** 9, p 747-774, (2009).

[23]. S.IRANI, , K.PRUHS: *Algorithmic problems in power management*; **SIGACT News**, 36, 2, p 63-76, (2003).

[24]. I. KARA, B.Y. KARA, M. KADRI YETIS. *Energy minimizing vehicle routing problem*. In A.Dress, Y.Xu, and B.Zhu, editors, **Combinatorial Optimization and Applications**, p 62– 71, Berlin, Heidelberg, (2007).

[25]. T.KELLY: *Generalized Knapsack solvers for multi-unit combinatorial auctions: Analysis and applications to computational resource allocation*; **Proc. International Workshop on Agent Mediated-Electronic Commerce AMEC** 2004, New-York p 73-86, (2004).

[26]. C.KOC, O.JABALI, J.MENDOZA, G. LAPORTE. *The electric vehicle routing problem with shared charging stations*. **International Transactions in Operational Research**, 26, (2018).

[27]. S.KOSUCH, A.LISSER: *On two-stage stochastic Knapsack problems*; **DAM** 159, p 1827-1841, (2011).

[28]. S.LAABADI, M.NAIMI, H.EL AMRI, B.ACHCHAB: *The 0/1 multidimensional Knapsack problem and its variants: a survey of models and heuristic approaches*; **American Journal of O.R** 8, p 395-439, (2018).

[29]. T.LUST, J.TEGHEM: *The multi-dimensional Knapsack problem: A survey and a new approach*; **International Transactions in Operational Research** 19, p 495-520, (2012).

[30]. S.MARTELLO, P.TOTH: *Knapsack problems: Algorithms and Computer implementations*; **John Wiley &Sons Ltd**, Chichester, (1990).

[31]. O.MASMOUDI, A.YALAOUI, Y.OUAZRNE, H.CHEHADE: *Lot Sizing in a multi-stage flow line production system with energy considerations*; **International Journal of Production Research** , 25 pages, (2016), DOI: 10.1080/00207543.2016.120660.

[32]. E.MEJIA-ALVAREZ, P.LEVNER, D.MOSSE: *Power optimized scheduling server for real time tasks*; **Proc. IEEE 8th Real Time Application Symposium** (**RTAS 2002**), San José, p 239-250, (2002).

[33]. G.MOUZON, M.B.YILDIRIM, J.TWOMEY: *Operational methods for minimizing energy consumption and manufacturing equipment*; **International Journal of Production Research** 45 (18-19), p 4247-4271, (2007).

[34]. C.H.PAPADIMITRIOU: *Computational Complexity*; **Addison Wesley**, Reading, MA, (1994).

[35]. K.PRUHS, G.J.WOEGINGER: *Approximation schemes for a class of subset selection problems*; **Theoretical Computer Sciences** 382, p 151-156, (2007).

[36]. J.PUCHINGER, G.R.RAIDL, U.PFERSCHY: *The multi-dimensional Knapsack problem: Structure and Algorithms*. **INFORMS Journal of Computing** 22, p 250-265, (2010).

[37]. M.RAYLAN, S.MATOS, Y.FRORA, and L.SATORY OCHI. *Green vehicle routing and scheduling problem with split delivery*. **Electronic Notes in Discrete Mathematics**, 69: 13 – 20, 2018. Joint, EURO/ALIO, International Conference 2018 on Applied Combinatorial Optimization, (2018).

[38]. F.SAHLING, L.BUSCHKUHL, H.TEMPELMEIER, S.HELBER: *Solving a multi-level lot sizing problem with multi-period setup carry-over via a fix and optimize heuristic*; **ECONSTOR, ZBW Diskussion Beitrag** 400, **Leibnitz Universitat Hannover**, 24 pages, (2008).

[39]. F.Z.SARGUT, H.E.ROMEIJN: *Capacitated production and subcontracting in a serial supply chain*; **IIE Transactions** 39 (11), p 1031-1043, (2007).

[40]. Y.SONG, C.ZHANG, Y.FANG: *Multiple multi-dimensional Knapsack problem and its applications to cognitive radio networks*; **Proc. IEEE Military Communications Conference (MILCOM 2008)**, San Diego, p 1-7, (2008).

[41]. F.TAILLANDIER, C.FENANDEZ, A.NDIAYE: *Real estate property maintenance optimization based on multi-objective multi-dimensional Knapsack problem*; **Computer Aided Civil Engineering** 32, p 227-251, (2017).

[42]. C.UMANS: *Hardness of approximating Σ^p_2 minimization problems*; **Proc; 40th Symposium on Foundations of Computer Sciences FOCS99**, Los Alamitos, p 465-474, (1999).

[43]. L.N.VICENTE, P.HCALAMAO: *Bi-level and multi-level programming : A bibliography review*; **Journal of Global Optimization** 5, p 291-306, (1994).