Communication Papers of the 19th Conference on Computer Science and Intelligence Systems (FedCSIS) pp. 131–138
DOI: 10.15439/2024F7665
ISSN 2300-5963 ACSIS, Vol. 41

Handling Lot Sizing/Job Scheduling Synchronization through Path Search Algorithms

Alain Quilliot and Aurélien Mombelli
LIMOS Lab.
UCA, CNRS and EMSE
Clermont-Ferrand, France
Email: alain.quilliot@uca.fr
LIMOS Lab.
Clermont-Ferrand, France

***Abstract*—We deal here with the synchronization of a resource production process and the consumption of those resources by machines used in order to run a set of jobs. Resources are produced by a *Resource Lot Sizer* player, under some economic costs and technical constraints. Both the *Resource Lot Sizer* and the *Job Scheduler* interact through *transfer transactions*, that involve specific constraints, making the synchronization between both processes become an issue. Resulting decision problem appears as a multi-stage *Lot Sizing* problem, arising in contexts where the resource is some renewable energy (hydrogen, electricity, ...) required by the jobs and stored inside tanks or batteries embedded into the machines (vehicles or robots). We first cast it into the MILP format, perform a structural analysis of the transfer transactions and handle resulting MILP_SLSS model through Branch and Cut before reformulating it as a path search problem set in a specific *Transfer* space and handling through a filtered adaptation of the A* algorithm.**

I. Introduction

LOT SIZING arises when one schedules production while taking care of avoiding strong activation costs (see [3]). A *Lot Sizing* strategy consists in concentrating production on well-chosen periods in order to minimize those activation costs. Resulting problems look like Knapsack problems involving a temporal dimension. One often handles them through dynamic programming, that may yields FPTAS: *Fully Polynomial Time Approximation Scheme*. Multi-dimensional Lot Sizing (see [**?**]) means a heterogeneous production and different kinds of capacity constraints (weight, volume, ...). Multi-level Lot Sizing means several players acting on a same system (a main resource producer and several subcontractors). Those players may be partially independent from each others, raising the collaborative issue and leading to cooperative game or multi-objective reformulations (see [2]). In any case, resulting problems are NP-Hard.

Now it may happen that such a process takes place in the context of an interaction between the *Lot Sizer* player, that is a *Resource Lot Sizer*, and a (several) *Job Scheduler(s)* who uses resources in order to run jobs (production or services) on parallel machines (vehicles or industrial robots). It is specifically the case when the resource is energy (power, hydrogen, biofuel, ...) loaded into batteries or tanks embedded into the machines (see [4]). In such a case, both the resource lot sizer and the job scheduler interact through *transfer transactions*, consisting in transferring resources (the energy) from the resource production plant into the storage facilities embedded into the machines. Both players must agree about the production periods i and the job dates t when the energy producer transfers some energy amount m to the job scheduler. Such a *transfer transaction* $\omega = (i, t, m)$ usually involves its own constraints and costs. According to this, we get a full description of our process if we know, besides the production vector z and the schedule of the jobs, the collection of all *transfer transaction* $\omega = (i, t, m)$ which make both players meet.

As the *Lot Sizing* problems, job scheduling problems have been widely investigated in the past (see [3]). It may arise in contexts related not only to energy management (see [2], [8]), but also to real time cooperation between sensors and robots, to the interaction between electric vehicles and recharge facilities (see [4], [5]). It raises the *centralized versus collaborative* issue (see [2]) since involved players may be independent. However, though both the *Lot Sizing* and job scheduling problems have been extensively studied, few authors addressed the synchronization issue that is at the core of the situation that we just described above (see [1]). Our main purpose here is to address it.

We stick here to the centralized paradigm and focus on the combinatorial issues related to synchronization and suppose that the processing order of the jobs $j = 0, \ldots, M-1$ on the machine have already been decided. We denote by **SLSS**: *Synchronized Lot Sizing/Scheduling* resulting problem that may be viewed as containing the *core* of the synchronization issue. We first make appear the central role played here by the *transfer transactions*: We check that, for any feasible solution of **SLSS**, related transfer transactions define a chain inside some partially ordered set. This leads us to next design a MILP: *Mixed Integer Linear Programming* setting of **SLSS** that is centered on those transfer transactions and involve complex *no-antichain* constraints. We check that, though those *no-antichain* constraints are exponential, they may be separated in polynomial time, opening the way to the handling of **SLSS** through Branch and Cut. Yet, this approach remains time consuming. Keeping on with the idea of taking advantage of the specific structure of the transfer transactions as a partially

ordered set, we reformulate **SLSS** problem as a path search problem inside some large size *Transfer* network. We deal with this path search reformulation while designing a *Path_SLSS* algorithm that proves itself far more efficient than the MILP setting and allows us to state that **SLSS** is pseudo-polynomial. So the paper is organized as follows. We describe in Section II the **SLSS** problem. In Section III, we study the structure of the transfer transactions and set a transfer transaction driven MILP formulation **MILP_SLSS** of **SLSS**, that involves *no-antichain* constraints. We explain in Section IV the way those *no-antichain* constraints may be separated, allowing us to deal with **MILP_SLSS** through branch and cut. Section V is devoted to our main contribution, that consists in a path search reformulation of **SLSS** and its more efficient handling through the design of a filtered A* like path search algorithm ([7]). This algorithm allows us to check that **SLSS** is pseudo-polynomial. We conclude with numerical experiments.

II. The Synchronized Lot Sizing/Scheduling Problem

We consider:

- **A Resource Lot Sizer:** The time horizon of the resource lot sizer is divided into N periods $i = 0, \ldots, N-1$, all with equal duration p. Thus the starting time of period i is equal to $p.i$. During a period i, this resource producer may be active or idle. If it is active, then it produces R_i resources, under a variable *running* cost $Cost_i^R$. If it is idle, then turning it into active induces an additional *activation* cost $Cost_i^A$. It is provided with a resource storage facility, with capacity C^P and initial load H_0^P. It cannot produce if resulting load exceeds C^P. This last hypothesis will be discussed in Section II.*B*. It must end with a load at least equal to H_0^P and while minimizing its production cost.
- **A Job Scheduler:** It must sequentially perform M jobs $j = 0, \ldots, M-1$ on a single machine, according to this order, within a time horizon $[0, N.p]$. Running a job j requires e_j resource units and t_j time units. It is provided with a resource storage facility, with capacity C^S and initial load equal to H_0^S. It must end with a load at least equal to H_0^S while minimizing that makespan, that means the ending time of job $M-1$.
- **Transfer Transactions:** Because of its limited storage capacity C^S, the job scheduler must periodically receive resources from the resource producer. Such a *transfer transaction* (see Figure 1) takes place during a whole period i, between the end of some job j and the beginning of its successor $j+1$, and involves the transfer of m resources. It is denoted as $\omega = (i, j, m)$, where $m \leq Inf(C^P, C^S)$. Performing $\omega = (i, j, m)$ requires a full period i, during which production is forbidden. It also requires from the job scheduler ϵ_j additional resources, τ_j additional time and a potential waiting time to be spent before the transfer transaction takes place. One must understand those requirements as related to a kind of detour (if the machine is a vehicle) or set up (if it is a robot). The consequence is that if $\omega = (i, j, m)$ defines a transfer transaction and if we denote by T_j the starting time of j, by V_j^S the resource load of the job scheduler at the time when j starts and by V_i^P the resource load of the resource lot sizer at the beginning of period i, then:
  - The starting time $p.i$ of period i must be at least equal to $T_j + t_j + \tau_j$ (augmented with a possible waiting time), while the ending time $p.(i+1)$ of i will coincide with the starting time T_{j+1} of job $j+1$.
  - The load V_{i+1}^P of the resource lot sizer at the beginning of period $i+1$ must be equal to $V_i^P - m$ and must be non negative.
  - The load V_{j+1}^S of the job scheduler at the beginning of job $j+1$ must be equal to $V_j^S - \epsilon_j + m$ must not exceed C^S; Its load $V_j^P - \epsilon_j$ at the beginning of period i must be non negative.

  Since those additional coefficients τ_j and ϵ_j most often corresponds to some kind of detour, we suppose that the following triangle inequalities hold for any $j \leq M-1$: $\tau_j \leq \tau_{j+1} + t_{j+1}$ and $\epsilon_j \leq \epsilon_{j+1} + e_{j+1}$.
  These inequalities mean that if the job scheduler has no time left for a transfer transaction at the end of job j, then keeping on with another job will not provide it with the missing time.
  In order to allow resources to be transferred before job 0 or after job $M-1$ we introduce a fictitious job -1, such that $t_{-1} = 0, e_{-1} = 0$, τ_{-1} and ϵ_{-1} being non null additional coefficients, and another fictitious job M, with $t_M = e_{-M} = \tau_M = \epsilon_M = 0$.

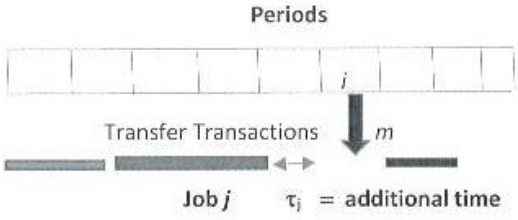

Fig. 1. A Transfer Transaction

Then solving the **SLSS:** ***Synchronized Lot Sizing/Scheduling*** problem means, in a natural way, computing the resource production periods, the starting times of the jobs, and the transfer transactions that link both players in such a way that:

- All jobs are done.
- Capacity constraints, job resource requirements and final state constraints are met.
- Some cost $\alpha.\sum_i PCost + \beta.T$ is minimized, where $PCost$ is the production cost, T is the ending time of the fictitious job M and α, β are *time versus money* coefficients.

III. A *Transfer Transactions* DRIVEN A MILP MODEL

As told in the introduction, transfer transactions (i, j, m) will be the leader component of any **SLSS** algorithmic solution.

A. The Transfer Transaction Partially Ordered Set

Let us first introduce some additional definitions:

Resource and Time Delays.

Let us recall that a transfer transaction $\omega = (i, j, m)$ starts at time $p.i$ and lasts one period. It starts just after the ending time of job j, augmented with delay τ_j and with a possible waiting time. Its ending time $p.(i+1)$ coincides with the starting time of $j+1$. Besides, it requires an additional resource ϵ_j, that the job scheduler consums just before the transaction takes place. A transfer transaction may take place just after the end of job $M-1$, at the very end of the process. It may also occur just before the beginning of job 0 or, in other words, between the fictitious job -1 provided with values ϵ_{-1} and τ_{-1} and job 0.

For any job pair (j_1, j_2) s.t. $0 \leq j_1 < j_2 \leq M-1$ we denote by $\mu(j_1, j_2) = \epsilon_{j_2} + \sum_{j_1+1 \leq j \leq j_2} e_j$ the amount of resources required between two transfer transactions respectively related to j_1 (just after j_1) and j_2 (just after j_2). We call this quantity the *resource_delay* between j_1 and j_2. This definition applies to $j_1 = -1$. By the same way, we set, for any $j_1 = -1, \ldots, M-1$:

- $\mu^{Start}(j_1) = (\sum_{j \leq j_1} e_j) + \epsilon_{j_1}$ that expresses the resource consumed between the beginning of the process and the first transfer transaction in case this transfer occurs just after j_1. We call it the *start_ resource_delay* induced by j_1. Notice that if $j_1 \neq -1$, then $\mu^{Start}(j_1) = \mu(-1, j_1)$.
- $\mu^{End}(j_1) = (\sum_{j > j_1} e_j)$ that expresses the resource consumed between the last transfer transaction in case this transfer transaction occurs just after j_1 and the end of the process. We call it the *end_resource_delay* induced by j_1. Notice that $\mu^{End}(j_1)$ might also be written $\mu(j_1, M)$ if we extend the definition of $\mu(j_1, j_2)$.

Considering the time instead of the resource, we set $\Delta(j_1, j_2) = \tau_{j_2} + \sum_{j_1+1 \leq j \leq j_2} t_j$, that means the time required between two transfer transactions respectively related to j_1 and j_2. This definition applies to $j_1 = -1$. We call this quantity the *time _delay* between j_1 and j_2. By the same way, we set, for any $j_1 = -1, \ldots, M-1$:

- $\Delta^{Start}(j_1) = (\sum_{j \leq j_1} t_j) + \tau_{j_1}$ that expresses the minimal time between the beginning of the process and the first transfer transaction in case this transaction occurs just after j_1. We call it the *start_time_delay* induced by j_1.
- $\Delta^{End}(j_1) = (\sum_{j > j_1} t_j)$ that expresses the time required between the last transfer transaction in case this transfer transaction occurs just after j_1 and the end of the process. We call it the *end_time_delay* induced by j_1.

We derive from those *time_delays* the following informations:

- We set $\pi_m(j) = \left\lceil \frac{\Delta^{Start}(j)}{p} \right\rceil$. We easily check that $\pi_m(j)$ means the first possible period for a transfer transaction involving a job $j \in \{-1, \ldots, M-1\}$.
- We set $\pi_M(j) = (N-1) - \left\lceil \frac{\Delta^{End}(j)}{p} \right\rceil$. We easily check that $\pi_M(j)$ means the last possible period for a transfer transaction involving $j \in \{-1, \ldots, M-1\}$.

A consequence of the triangle inequalities $\tau_j \leq \tau_{j+1} + t_{j+1}$ is that the interval $\{\pi_m(j), \ldots, \pi_M(j)\}$ provides us with the periods when a transfer transaction involving j may occur.

Also:

- For any j such that $M-1 \geq j \geq 0$, we denote by $\Phi(j)$ the unique job j_1 (it may not exist) such that $j < j1 \leq M-2$ and $\mu(j, j_1) \leq C^S$ and that $\mu(j, j_1+1) > C^S$.
- By the way, we denote by $\Phi(-1)$ the largest job j_1 such that $\mu^{Start}(j_1) \leq H_0^S$ and by $\Phi(M)$ the smallest j_1 such that $\mu^{End}(j_1) \leq C^S - H_0^S$.

A consequence of the inequalities $\epsilon_j \leq \epsilon_{j+1} + e_{j+1}$ is that if $\Phi(j)$ exists, then it tells us that at least one transfer transaction must take place that involves j_1 belonging to the period interval $\{j+1, \ldots, \Phi(j)\}$. If such a transaction were not existing, then moving ahead of $\Phi(j)$ or moving backward from j would not allow make appear any feasible transfer transaction, and the whole process would become infeasible.

Finally we denote by $C^{TR} = min(C^P, C^S)$ the maximal value m that may be involved into a transfer transaction.

The *Transfer Transaction* Partially Ordered Set $(\Omega, \ll)$.

We define Ω as the set of all (period/job) pairs (i, j) such that $\tau_m(j) \leq i \leq \tau_M(j)$, augmented with 2 fictitious pairs $Source$ and $Sink$. It contains all pairs (i, j) that may be involved into a transfer transaction.

We set $(i_1, j_1) \ll (i_2, j_2)$ if and only if a transfer transaction related to period i_2 and job j_2 can be preceded by another transfer transaction between period i_1 and station j_1 in the sense of the time delays, that means if $j_2 > j_1$ and $(i_2 - i_1 - 1).p \geq \Delta(j_1, j_2)$. We set $Source \ll (i, j)$ and $(i, j) \ll Sink$ for any (i, j). We easily check that this relation $\ll$ is transitive and anti-reflexive and so defines a partial order relation on Ω.

The order relation $\ll$ induces on the set Ω an oriented graph structure $G^{\ll}$. Any arc $((i_1, j_1), (i_2, j_2))$ of $G^{\ll}$ may be provided with a length $L^{\ll}((i_1, j_1), (i_2, j_2))$ as follows:

- If both (i_1, j_1) and (i_2, j_2) are different from $Source$ and $Sink$, then $L^{\ll}((i_1, j_1), (i_2, j_2)) = \mu(j_1, j_2)$;
- If $(i_1, j_1) = Source$ then $L^{\ll}((i_1, j_1), (i_2, j_2)) = \mu^{Start}(j_2)$;
- If $(i_2, j_2) = Sink$ then $L^{\ll}((i_1, j_1), (i_2, j_2)) = \mu^{End}(j_1)$.

Lemma 1: *Given a feasible solution of* **SLSS**. *Then the transfer transactions related to this solution define a path* Γ *in the acyclic graph* $G^{\ll}$, *that connects* $Source$ *to* $Sink$ *and is such that, for any arc* $((i_1, j_1), (i_2, j_2))$ *of* Γ*:*

- *If* (i_1, j_1) and (i_2, j_2) *are both different from* $Source$ *and* $Sink$, *then:* $L^{\ll}((i_1, j_1), (i_2, j_2)) \leq C^S$;
- *If* $(i_1, j_1) = Source$ *then* $L^{\ll}((i_1, j_1), (i_2, j_2)) \leq H_0^S$;

- *If* $(i_2, j_2) = Sink$ *then* $L^{\ll}((i_1, j_1), (i_2, j_2)) \leq C^S - H_0^S$.

Proof: TTw consecutive transfer transactions $(i_1, j_1, m_1), (i_2, j_2, m_2)$ involved into a feasible solution of **SLSS** define an arc in the sense of $(\Omega, \ll)$ because the time between the end of i_1 and the beginning of i_2 must be large enough in order to make possible processing the jobs $j_1 + 1, \ldots, j_2$. The bound imposed to the length of $((i_1, j_1), (i_2, j_2))$ means that the resources needed in order to process those jobs under the additional resource ϵ_{j_2} requirement cannot exceed the storage capacity C^S. □

*B. A MILP Formulation of **SLSS***

Let us recall that a $\ll$-*Antichain* of the partially ordered set $(\Omega, \ll)$ is any subset of Ω made of pairs (i, j) pairwise incomparable in the sense of $\ll$. Then we set **SLSS_MILP** by first introducing the following variables:

- **Transfer Transaction Variables**
  - $\{0, 1\}$-valued vector $U = (U_{i,j}, i = 0, \ldots, N-1, j = -1, \ldots, M-1)$: $U_{i,j} = 1$ means that some transfer transaction (i, j, m) is part of the solution.
  - Rational non negative vector $m = (m_{i,j}, i = 0, \ldots, N-1, j = -1, \ldots, M-1)$: If $U_{i,j} = 1$ then $m_{i,j}$ means related resources.
- **Resource Lot Sizing Variables**
  - $\{0, 1\}$-valued vector $y = (y_i, i = 0, \ldots, N-1)$: $y_i = 1$ means that the production is activated at the beginning of period i.
  - $\{0, 1\}$-valued vector $z = (z_i, i = 0, \ldots, N-1)$: $z_i = 1$ means the production active at period i.
  - $\{0, 1\}$-valued vector $\delta = (\delta_i, i = 0, \ldots, N-1)$: $\delta_i = 1$ means a transfer transaction at period i.
- **Job Variables**
  - T: $T =$ the ending time of the job M.
  - $\{0, 1\}$-valued vector $\gamma = (\gamma_j, j = -1, \ldots, M-1)$: $\gamma_j = 1$ means that a transfer transaction takes place at the end of job j. (If $j = -1$ then it means that a transfer transaction takes place just before job 0).

Then, the constraints of the **SLSS_MILP** come as follows:

- **Structural Constraints**

$$\text{Minimize} \sum_i (Cost_i^R.z_i + Cost_i^A.y_i) + \alpha.T \quad (*)$$

$$\forall j: \quad T \geq \sum_i U_{i,j}.(p(i+1) + \Delta^{End}(j)) \quad (C.1)$$

$$y_0 - z_0 \geq 0 \quad (C.2)$$

$$\forall i \geq 1: \quad y_i \geq z_i - z_{i-1} \quad (C.3)$$

$$\forall i \geq 0: \quad z_i + \delta_i \leq 1 \quad (C.4)$$

$$\forall i: \quad \delta_i = \sum_j U_{i,j} \quad (C.5)$$

$$\forall j: \quad \gamma_j = \sum_i U_{i,j} \quad (C.6)$$

$$\forall (i,j) \text{ s. t.} (i > \tau_M(j)) \vee (i < \tau_m(j)): \quad U_{i,j} = 0 \quad (C.7)$$

$$\forall A, A \ll\text{-Antichain}: \quad \sum_{(i,j) \in A} U_{i,j} \leq 1$$

$$(C.8 : \textit{No-Antichain Constraints})$$

$$\forall j \text{ s. t.} -1 \leq j \leq M-1:$$

$$\sum_{k \in \{j+1, \ldots, \Phi(j)\}} \gamma_k \geq 1 \quad (C.9)$$

$$\sum_{k \in \{-1, \ldots, \Phi(-1)\}} \gamma_k \geq 1 \quad (C.10)$$

$$\sum_{k \in \{\Phi(M), \ldots, M-1\}} \gamma_k \geq 1 \quad (C.11)$$

$$T \geq \sum_j (t_j + \tau_j.\gamma_j) \quad (C.12)$$

$$\forall j: \quad H_0^P + \sum_{k < \tau_M(j)} z_k.R_k \geq \sum_{k \leq j} (\gamma_k.\epsilon_k + e_k) - H_0^S \quad (R.1)$$

- **Resource Amount Constraints**

$$\forall i, j: \quad m_{i,j} \leq U_{i,j}.C^{TR} \quad (TL.1)$$

$$\forall i: \quad \sum_{j,k \leq i} m_{k,j} - \sum_{k<i} z_k.R_k \leq H_0^P \quad (TL.2)$$

$$\forall i: \quad \sum_{k \leq i} z_k.R_k - \sum_{j,k \leq i} m_{k,j} \leq C^P - H_0^P \quad (TL.3)$$

$$\sum_i z_i.R_i - \sum_{i,j} m_{i,j} \geq 0 \quad (TL.4)$$

$$\forall j \text{ s. t. } 0 \leq j \leq M-1:$$

$$\sum_{i,k<j} m_{i,k} + H_0^S \geq \sum_{k \leq j} (e_k + \gamma_k.\epsilon_k) \quad (TL.5)$$

$$\sum_{i,j} m_{i,j} \geq \sum_{j \leq M-1} (e_j + \gamma_j.\epsilon_j) \quad (TL.6)$$

$$\forall j \text{ s. t. } -1 \leq j \leq M-1:$$

$$\sum_{i,k \leq j} m_{i,k} \leq C^S - H_0^S + \sum_{k \leq j} (e_k + \gamma_k.\epsilon_k) \quad (TL.7)$$

Theorem 1. *The MILP program **SLSS_MILP** model solves the **SLSS** problem in an exact way.*

Proof. (*) clearly means the objective function of **SLSS**, while one easily checks that (C.1, C.2, C.3, C.4, C.5, C.6, C.7, C.8, C9, C10, C11, TL.1, TL.2, TL.3, TL.6) are necessary. (C.12): T is at least equal to the running times of the jobs and the transfer transactions. (R.1): For any job j, the resources produced before $j+1$ augmented with the initial resources, must be enough for jobs 0 to $j+1$. (TL.4): We must globally produce at least as much as we transfer. (TL.5): For any j, the resources needed in order to perform jobs $0, \ldots, j$ together with the related transfer transactions must not exceed available resources (the triangle inequalities are involved). (TL.7): For any j, the resources of the job scheduler after running j must not exceed C^P.

Conversely, we get sufficiency by first checking that constraints (C.5, ..., C.8) imply that the transfer transactions deriving from U define a chain in $(\Omega, \ll)$. Constraints (C.9, C.10, C11) make this chain consistent with Lemma 1. Let us denote it by $\Gamma = \{(i_1, j_1), \ldots, (i_Q, j_Q)\}$ and let us introduce variables $T_j, V_j^S, j = 0, \ldots, M-1, \hat{V}_q^S, q = 1, \ldots, Q, V_i^P, i = 0, \ldots, N-1, \hat{V}_q^P, q = 1, \ldots, Q$, respectively representing the starting time of job j, the resources stored by the job scheduler at time T_j, the resources stored by the job scheduler just before the q^{th} transfer transaction takes place, the resources stored by the resource lot sizer at the beginning of period i and the resources stored by the resource lot sizer just before the q^{th} transfer transaction takes place (at the beginning of period i_q). Then it is enough to follow Γ in order to provide those variables with values defining a feasible schedule. □

IV. A Branch and Cut for the SLSS_MILP model

Applying a MILP library to **SLSS_MILP** means designing a procedure in order to separate the *No_Antichain* constraints (C.8), that means in order to dynamically generate and insert new *No_Antichain* constraints as soon as a current solution of the rational relaxation of **SLSS_MILP** appears, that does not meet those constraints. Let U be some rational (or integral) vector satisfying (C4, C5, C6, C7). Separating (C.8) means searching for an antichain A in $(\Omega, \ll)$ such that $\sum_{(i,j)\in A} U_{i,j} > 1$. Theoretically (see [6]) it can be done in polynomial time by application of a feasible network flow procedure. However, the arcs of $G^{\ll}$ are too many for an efficient application of such a separation procedure. Instead, we proceed in a heuristic way, through partial tree search. Related *backtracking* nodes correspond to sequences $\{(i_1, j_1), \ldots, (i_k, j_k)\}$ ordered according to decreasing $U_{i,j}$ values and whose elements define an antichain in Ω. In order to speed the process, we arbitrarily impose an upper bound on the number of backtracking nodes created this way and use this upper bound as a control parameter. If U is integral, then we only need to scan the transfer transactions and check that 2 consecutive transfer transactions agree with $\ll$.

V. Handling SLSS as a Path Search Problem

Let us come now to our main contribution. We are going to show that **SLSS** may be reformulated as a path search problem in an acyclic large state network. This will allow us to handle it while adapting the well-known A* algorithm (see [7]) for the computation of robot trajectories inside state spaces. This approach will prove itself to be far more efficient that the MILP based approach.

A. The SLSS Path Search Formulation

We saw in Section III.*A* that the transfer transactions involved in any feasible **SLSS** solution define in the graph $G^{\ll}$ a path Γ from $Source$ to $Sink$ such that if (i_1, j_1) and (i_2, j_2) are consecutive in Γ, then: (P0)

- If $(i_1, j_1) \neq Source$ and if $(i_2, j_2) \neq Sink$ then $\mu(j_1, j_2) \leq C^S$
- If $(i_1, j_1) = Source$ then $\mu^{Start}(j_2) \leq H_0^S$
- If $(i_2, j_2) = Sink$ then $\mu^{End}(j_1) \leq C^S - H_0^S$

Such a path $\Gamma = \{Source, (i_1, j_1), (i_2, j_2), \ldots, (i_Q, j_Q), Sink\}$ does not fully characterize a **SLSS** feasible solution. But we are going to show that it will do it as soon as we are able to provide it with well-fitted sequences $w = \{w_q, \quad q = 0, 1, \ldots, Q, Q+1\}$ and $W = \{W_q, q = 0, 1, \ldots, Q\}$, with the meaning:

- For any $q = 1, \ldots, Q$, w_q means the resources globally stored together by the resource lot sizer and the job scheduler at the end of period i_q. For any $q = 1, \ldots, Q-1$, W_q denotes the resource produced between period i_q and period i_{q+1}.
- $w_0 = H_0^P + H_0^S =$ the initial resources and W_0 means the resources produced before period i_1. $W_Q =$ the resources produced after period Q.
- $w_{Q+1} \geq H_0^P + H_0^S$ means the final global resources.

Then we may characterize the **SLSS** solutions according to the following statements:

Theorem 2. *Given a path* $\Gamma = \{Source, (i_1, j_1), (i_2, j_2), \ldots, (i_Q, j_Q), Sink\}$ *that meets (P0), together with non negative values* $w_q, q = 0, 1, \ldots, Q, Q+1$ *and* $W_q, q = 0, 1, \ldots, Q$. *They may be extended into a feasible solution of* ***SLSS*** *iff:*

1) $w_1 = w_0 + W_0 - \mu^{Start}(j_1)$;
   $w_0 = H_0^P + H_0^S$;
   $W_0 \leq C^P - H_0^P$;
   W_0 *is a feasible production for periods in* $\{0, \ldots, i_1 - 1\}$. *(P1)*
2) *For any* $q = 1, \ldots, Q-1$: *(P2)*
   - $W_q \leq min(C^P, C^S + C^P - w_q)$ *is a feasible production for periods in* $\{i_q + 1, \ldots, i_{q+1} - 1\}$;
   - $w_{q+1} = w_q + W_q - \mu(j_q, j_{q+1})$;
   - $\mu(j_q, j_{q+1} \leq Inf(w_q, C^S)$.
3) $W_Q \leq min(C^P, C^S + C^P - w_q)$ *is a feasible production for periods in* $\{i_Q + 1, \ldots, N-1\}$;
   $w_{Q+1} = (w_Q + W_Q - \mu^{End}(j_Q) \geq H_0^P + H_0^S$; *(P3)*

Proof. Lemma 1 tells us that we may derive from path Γ starting times values $T_j, j = 0, \ldots, M-1$ for the jobs j, that are consistent with the time requirements of **SLSS**. Let us suppose that we know the values $\hat{w}_q^S, q = 1, \ldots, Q, \hat{w}_q^P, q = 1, \ldots, Q$ denoting the resources available for respectively the job scheduler and the resource lot sizer just after the q^{th} transfer transaction takes place (at the end of period i_q). Then we easily deduce the resources $V_j^S, j = 0, \ldots, M-1$ available for the job scheduler at time T_j, and the resources $V_i^P, i = 0, \ldots, N-1$ available for the resource producer at time $p.i$. So the key point becomes distributing global resource quantities w_q among the job scheduler and the resource lot sizer, and get quantities $\hat{w}_q^S, q = 1, \ldots, Q, \hat{w}_q^P, q = 1, \ldots, Q$ that mean the resources available for respectively the job scheduler and the resource lot sizer at the end of period i_q, in such a way that resulting quantities $V_j^S, j = 0, \ldots, M-1$

and $V_i^P, i = 0, \ldots, N-1$ meet the resource requirements of **SLSS**. We get it by applying the following rule: For every q, we complete the pair (i_q, j_q) with a transfer value m_q in such a way that either we completely fill the storage device of the job scheduler ($\hat{w}_q^S = C^S$) or, if it is not possible, we make the storage facility of the resource lot sizer become empty ($\hat{w}_q^P = 0$). Then, it becomes a matter of routine to check that resulting $V_j^S, j = 0, \ldots, M-1$ and $V_i^P, i = 0, \ldots, N-1$ meet the resource requirements of **SLSS**. □

A Path Search Formulation of SLSS.

Theorem 2 suggests us the construction of the following *Augmented Transfer* oriented graph $G^{Augment,\ll}$:

- The nodes of $G^{Augment,\ll}$ are pairs (x, w), where x is a node of $G^{\ll}$ and w is a resource value such that $0 \leq w \leq C^S + C^P$; We define the node $(Source, H_0^S + E_0^P)$ as a *source* node and any node $(Sink, w \geq H_0^S + H_0^P)$ as a *sink* node.
- We say that $((x_1, w_1), (x_2, w_2))$ defines an arc of $G^{Augment,\ll}$ if (x_1, x_2) is an arc of $G^{\ll}$ that meets (P0) and if w_2 may be written $w_2 = w_1 + W - \mu(j_1, j_2)$, W being a feasible production for periods in $i_1 + 1, \ldots, i_2 - 1$, in a way that meets (P1, P2, P3) of Theorem 2.
- The cost $Cost^{Augment}((x_1, w_1), (x_2, w_2))$ of such an arc is $\alpha.PCost(W) + \beta.(i_2 - i_1)$, where $PCost(W)$ is the production cost induced by W. In case $x_1 = Source$ or $x_2 = Sink$, then this formula has to be respectively adapted as $\alpha.PCost(W) + \beta.\Delta^{Start}(j_2)$ and $\alpha.PCost(W) + \beta.\Delta^{End}(j_1)$.

Then we get a **SLSS Path Reformulation**: {Compute a shortest path (in the sense of $Cost^{Augment}$) in $G^{Augment,\ll}$ from the *source* node to the *sink* nodes.}

B. An A Based Algorithm*

Above **SLSS** path reformulation suggests us to deal with **SLSS** while relying on a standard path search algorithm for acyclic graphs. However, we must take care on the 2 following specific features of our path model:

1) Every time we are working with some node (i_1, j_1, w_1) of $G^{Augment,\ll}$, we must generate arcs $((i_1, j_1, w_1), (i_2, j_2, w_2))$, together with values $W = w_2 - w_1 + \mu(j_1, j_2)$ and related cost values $PCost(W)$. But computing these cost values requires the resolution of some local lot sizing problem.
2) Depending on the context, values w may be large, possibly infinite. So we need filtering devices.

Computing the Cost Values through a Pre-Process.

We deal with the first issue while performing a dynamic programming pre-process that yields a table TAB, indexed on the pair $i_1, i_2, -1 \leq i_1 < i_2 \leq N$. For any such a pair, $TAB[i_1, i_2]$ contains a list of 3-uples $(W, PCost(W), Sol)$ where W means the resources produced during periods $i = i_1 + 1, \ldots, i_2 - 1$, $PCost(W)$ means their optimal production cost, and Sol some related production schedule. This dynamic programming pre-process is implemented according to a backward driven strategy: It involves a main loop indexed on the 2-uples (i_2, w_2^P) and an internal loop indexed on the 2-uples (i_1, w_1^P) such that $i_1 < i_2$, and works in pseudo-polynomial time. We may speed it by noticing that if $i_2 \leq N-2$, then we may restrict ourselves to pairs $i_1, i_2)$ such that it is possible to compute j_1, j_2 with $i_1 \geq \tau^m(j_1)$ and $i_2 \leq \tau M(j_2)$. So, every time we generate a decision value W, we get its cost value $PCost(W)$ through a direct access to the table TAB. We notice that if C^S and C^P are bounded by polynomial functions of N and M then the construction of TAB can be performed in polynomial time, which also means that in the general case, this construction can be performed in pseudo-polynomial time.

Introducing Filtering Devices: A Lower Bound.

As for the second issue, we may reduce the number of nodes explored during our path search by performing some kind of rounding (for instance modulo the first k bits, as usually done in order to design FPTAS algorithms). But the most natural way is to rely on a lower bound procedure, that, with some node (i_1, j_1, w_1) of $G^{Augment,\ll}$ is going to associate a lower bound $LB(i_1, j_1, w_1)$ of a shortest path from (i_1, j_1, w_1) to the *sink* nodes in $G^{Augment,\ll}$. We get such a lower bound by first computing a lower bound $Trans_{j_1}^{Min}$ on the number of transfer transactions that will remain to be performed after period i_1. We see that we may define $Trans_{j_1}^{Min}$ as follows:

$$Trans_{j_1,w_1}^{Min} = \lceil \frac{(H_0^S - w_1 + \sum_{j>j_1} e_j)}{C^{TR}} \rceil,$$

Next we notice that we get a lower bound on the additional time $T\_Add_{j_1,w_1}^{Min}$ that will to be spent by the job scheduler because of those transfer transactions by setting:

- $\tau_{j_1}^{Min}$ = the smallest value $\tau_j, j \geq j_1 + 1$;
- $T\text{-}Add_{j_1,w_1}^{Min} = \tau_{j_1}^{Min} \cdot Trans_{j_1,w_1}^{Min}$.

We deduce a lower bound T_{j_1,w_1}^{Min} on the time that the job scheduler will have to spend after the end of period i_1 before achieving its own process by setting:

$$T_{j_1,w_1}^{Min} = (\sum_{j>j_1} t_j) + T\text{-}Add_{j_1,w_1}^{Min}.$$

By the same way, we check that achieving this process will require during the periods $i_1, \ldots, N-1$ the production of at least W_{j_1,w_1}^{Min} resource, where W_{j_1,w_1}^{Min} is defined by:

- $\epsilon_{j_1}^{Min}$ = the smallest value $\epsilon_j, j \geq j_1 + 1$;
- $W_{j_1,w_1}^{Min} = H_0^P + H_0^S + \mu^{End}(j_1) + \epsilon_{j_1}^{Min} . Trans_{j_1,w_1}^{Min} - w_1$.

Then we may retrieve from $TAB[i_1, N]$ a value $PCost_{i_1,j_1,w_1}^{Min}$ equal to the smallest cost value related to some production W such that $W \geq W_{j_1,w_1}^{Min}$;

Lemma 3: $LB(i_1, j_1, w_1) = \beta \cdot T_{j_1,w_1}^{Min} + \alpha.PCost_{i_1,j_1,w_1}^{Min}$ *is a lower bound for the cost of a shortest path from* (i_1, j_1, w_1) *to the sink nodes in* $G^{Augment,\ll}$.

Proof. It derives in a straightforward way from the construction of the quantities T_{j_1,w_1}^{Min} and $PCost_{i_1,j_1,w_1}^{Min}$. □

Exact and Heuristic Dominance Rules.

We enhance our algorithm by introducing dominance rules:

- **Exact Dominance Rule:** If at some time during the search process we deal with two nodes $\sigma_1 = (i_1, j_1, w_1)$ and $\sigma_2 = (i_2, j_2, w_2)$ of $G^{Augment,\ll}$ such that $(i_1, j_1, w_1) = (i_2, j_2, w_2)$ and $Cost^{G\_Augment}(\sigma_1) + LB(\sigma_1) \leq Cost^{G\_Augment}(\sigma_2) + LB(\sigma_2)$, then we may *kill* σ_2.
- **Heuristic Dominance Rule:** We may reinforce above rule by setting that $\sigma_1 = (i_1, j_1, w_1)$ dominates $\sigma_2 = (i_2, j_2, w_2)$ if:
  - $Cost^{G\_Augment}(\sigma_1) + LB(\sigma_1) \leq Cost^{G\_Augment}(\sigma_2) + LB(\sigma_2)$
  - $i_1 \leq i_2$, $j_2 \leq j_1$ and $w_1 \leq w_2$.

  If at some time during the search process we deal with two nodes $\sigma_1 = (i_1, j_1, w_1)$ and $\sigma_2 = (i_2, j_2, w_2)$ of $G^{Augment,\ll}$ such that σ_1 dominates σ_2 , then we *kill* σ_2. Yet this rule is only a heuristic dominance rule. Yet, numerical experiments will show its efficiency.

The Algorithm *A*_SLSS*: So we design our algorithm *A*_SLSS* as an adaptation of the well-known *A** (see [7]) algorithm for path search in very large networks. At any time during the *A*_SLSS* resolution process, we are provided with an expansion list LS of nodes $\sigma = (i, j, w)$of $G^{Augment,\ll}$, given together with related cumulative costs $Cost^{G\_Augment}(\sigma)$ and future cost estimations $LB(\sigma)$. Those nodes are ordered according to increasing $Eval(\sigma) = Cost^{G\_Augment}(\sigma) + LB(\sigma)$ estimation values. Then we pick up the first element $\sigma_1 = (i_1, j_1, w_1)$ in LS, called the *pivot* node, and we expand it: we generate all decisions (i_2, j_2, W) such that $(i_1, j_1) \ll (i_2, j_2)$, (P0) holds and $W \in TAB[i_1, i_2]$ that are valid in the sense that they meet (P1, P2, P3) of Theorem 2. For every such a decision, we generate resulting state $\sigma_2 = (i_2, j_2, w_2)$ and insert it into LS while meeting the dominance rules (depending on the rules, it will yield an exact or heuristic algorithm). We stop when the *pivot* state σ is a *sink* node.

Resulting *A*_SLSS* algorithm may be summarized:

Algorithm *A*_SLSS*

Initialization:

- $LS = \{source\}$, $sink$ being the *s*ource node of $G^{Augment,\ll}$, provided with $Eval(sink) = LB(sink)$ estimation value; $NotStop$;
- $Curr\_Sol$ = current partial solution set = Nil;

While $NotStop$ and $LS \neq Nil$ do

1) Denote by $Pivot$ the head of LS;
2) Remove it from $LS$and insert it into $Curr\_Sol$;
3) If $Pivot = (i_1, j_1, w_1)$ is a *sink* node then $Stop$
   Else

   Generate all valid decisions $Dec = (i_2, j_2, W)$ that apply to $Pivot$;
   For any such a decision Dec do
   - Compute resulting node $\sigma_2 = (i_2, j_2, w_2)$, together with its value $Eval(i_2, j_2, w_2)$;
   - If σ_2 is dominated by no node σ_0 in LS then insert it into LS, while keeping LS from containing a node σ_0 dominated by σ_2;

   If $Stop$ then Retrieve from $Pivot$ and $Curr\_Sol$ a full path Γ solution of **SLSS**;

Depending on the dominance rules that we apply here, we obtain two algorithms *A*_SLSS* and *Heur_A*_SLSS* .

Theorem 3. *When implemented with the weak dominance rules, above algorithm A*_SLSS solves **SLSS** in an exact way. In any case, it works in pseudo-polynomial time.*

Proof. The first part of this statement derives from Theorem 2 in a straightforward way. This algorithm is nothing more than the *A** algorithm applied to the state network whose nodes are all states $\sigma = (i, j, w)$, and arcs corresponds to transitions $(\sigma 1 = (i_1, j_1, w_1) \leftarrow (\sigma 2 = (i_2, j_2, w_2))$ according to decisions (i_2, j_2, W). As for the second part, we see that if C^S and C^P are bounded by polynomial functions of N and M then the number of nodes of the graph $G^{Augment,\ll}$ is also bounded by a polynomial function of N and M, while TAB may be computed in pseudo-polynomial time. We conclude. □

VI. Numerical experiments

Purpose: Evaluating the behavior of the A^* algorithm, with respect to **SLSS_MILP**, considered as a benchmark.

Technical Context: Algorithms were implemented in C++ on an Intel i5-9500 CPU at 4.1GHz. CPU times are in seconds. We used the CPLEX20 library for the MILP models.

A. Instance generation

Production and consumption coefficients: In order to mimic what may be a power market, we cluster production periods into $\#SP$ super-periods of same length. Each super-period is assigned symbolic mean production and cost values $\overline{R_{cl}}$, $\overline{Cost_{cl}}$ in $\{Low, Medium, High\}$. Then, integral production and variable cost values are randomly generated for every period according to those mean super-period values. We generate activation costs in such a way that the activation cost represents around a third on the global production cost.

Storage capacities and scaling coefficients α, β: In order to control the relation between the number of transfer transactions and the number of activation decisions, we impose the quotient $\frac{C^P}{C^S}$ to remain inside an interval $[0.5, 3]$. We do in such a way that $\frac{\mu(-1,M)}{C^{TR}}$ evolves like M, so that we may control the number of jobs between transfer transactions (in average close to 5). We do in such a way that the respective weights of the production and the scheduling parts of the global cost remain integral and comparable.

Tables I and II present a package of 12 instances.

B. Outputs

- **Tables III and IV** : This table is devoted to the **SLSS_MILP** model. It provides the objective value obj of **SLSS_MILP**, its linear relaxation $relax$, the number $A - Cuts$, of *No-Antichain* cuts generated during the process, related CPU times CPU.

TABLE I
INSTANCE PARAMETERS TABLE

id	1	2	3	4	5	6
N	20	30	40	60	70	80
M	10	10	15	20	20	25
p	2	3	4	2	3	4
$\#SP$	2	2	3	4	5	5

TABLE II
INSTANCE PARAMETERS TABLE

id	7	8	9	10	11	12
N	100	110	120	140	150	160
M	30	30	35	40	40	40
p	2	3	4	2	3	4
$\#SP$	5	5	5	4	5	5

- **Table V and VI** : This table is devoted to the path search approach. It provides resulting number TT of transfer transactions, together with CPU times $Path - CPU$ induced by the A^* algorithm. It also provides the same values $HF - TT$ and $HF - Path - CPU$, together with the cost value $HF - obj$, obtained by introducing the strong (heuristic) dominance rules of Section V, that means by applying the *Heur_A*_SLSS* Algorithm.

Comments: The MILP model is time consumming, due to the gap induced by the relaxation of its integrality constraints. The A^* algorithm significantly outperforms the MILP model. The heuristic dominance rule of Section V.*B* induces a very small gap with respect to optimality (it reach optimality in 10 among the 12 instances, and the gap for the 2 remaining instances 8 and 12 hardly reaches 2%, while speeding in average the search process by 40%.

VII. FUTURE WORK

Future research will be oriented towards: 1) the **SLSS** extensions that make the parallel machine scheduling decisions be part of the problem; 2) the collaborative issue, when several *job schedulers* interact with the *Lot Sizer* player; 3) the management of uncertainty.

TABLE III
MILP RESOLUTION

id	1	2	3	4	5	6
obj	186	202	238	386	402	456
$A - Cuts$	158	179	326	595	554	818
$relax$	103.6	131.8	147.8	270.8	285.6	321.6
CPU	31.8	20.0	32.6	998	1997	1024

TABLE IV
MILP RESOLUTION

id	7	8	9	10	11	12
obj	558	596	650	684	758	781
$A - Cuts$	357	882	1071	17431	1928	2235
$relax$	376.6	388.2	360.0	452.5	564.0	536.1
CPU	3079	2889	5906	10058	11647	7945

TABLE V
A* ALGORITHM RESOLUTION

id	1	2	3	4	5	6
TT	3	4	3	5	7	6
$Path - CPU$	2.1	3.2	1.9	15.8	57.4	45.6
$HF - obj$	186	202	238	386	402	456
$HF - TT$	3	4	3	5	7	6
$HF - Path - CPU$	1.8	2.6	1.5	10.9	43.5	25.6

TABLE VI
A* ALGORITHM RESOLUTION

id	7	8	9	10	11	12
TT	5	8	6	9	7	10
$Path - CPU$	38.3	148.7	61.8	598.9	265.6	677.7
$HF - obj$	558	608	650	684	758	802
$HF - TT$	5	9	6	9	7	10
$HF - Path - CPU$	28.1	83.2	42.6	395.8	137.4	400.6

ACKNOWLEDGMENT

Present work was funded by French ANR: National Agency for Research, Labex IMOBS3, and PGMO Program.

REFERENCES

[1] F. Bendali, J. Mailfert, E. Mole-Kamga, A. Quilliot, H. Toussaint, *Pipelining dynamic programming process in order to synchronize energy production and consumption*, Proc. 2020 FEDCSIS WCO Conf., p 303-306, 2020. doi.org/10.15439/978-83-955416-7-4.

[2] Biel K., Glock C. H.: Systematic literature review of decision support models for energy-efficient production planning. *Computers and Industrial Engineering*, 101, pp. 243-259, (2016). https://doi.org/10.1016/j.cie.2016.08.021.

[3] Clark A. Almada-Lobo B., Almeder C.: , J.: Lot sizing and scheduling: Industrial extensions and research opportunities. *International Journal of Production Research* 49-9, p 2457-2461 (2011). https://doi.org/10.1080/00207543.2010.532908.

[4] Erdelic T., Caric T., Lalla-Ruiz E.: A Survey on the Electric Vehicle Routing Problem: Variants and Solution Approaches.*Journal of Advanced Transportation* Volume 2019. Article ID 5075671; https://doi.org/10.1155/2019/5075671.

[5] S. Fidanova, O. Roeva, M. Ganzha, *Ant colony optimization algorithm for fuzzy transport modelling*, Proc. 2020 FEDCSIS WCO Conference, p 237-240, 2020. doi.org/10.15439/978-83-955416-7-4

[6] Franck A.: On chain and antichain families of a partially ordered set. *Journal of Combinatorial Theory. Series B* 29-2, p 176-184 (1980). https://doi.org/10.1016/0095-8956(80)90079-9.

[7] Hart P. E., Nilsson N. J., Bertram R.: A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 4-2, p 100-107 (1968). https://doi.org/10.1109/TSSC.1968.300136.

[8] Irani S, Pruhs K.: Algorithmic problems in power management. *ACM SIGACT News* 36-2, p 63-76 (2005). DOI:10.1145/1067309.1067324.

[9] K. Stoilova, T. Stoilov, *Bi-level optimization application for urban traffic management*, Proc. 2020 FEDCSIS WCO Conf., p 327-336, 2020. doi.org/10.15439/978-83-949419-5-6.