Minimizing makespan under data prefetching constraints for embedded vision systems: a study of optimization methods and their performance

Khadija Hadj Salem, Vincent Jost, Yann Kieffer, Luc Libralesso, Stéphane Mancini

Oper Res Int J manuscript No.
(will be inserted by the editor)

Minimizing makespan under data prefetching constraints for embedded vision systems: a study of optimization methods and their performance

Khadija Hadj Salem[1] 0000-0003-3739-2899 · Vincent Jost[2] · Yann Kieffer[3] · Luc Libralesso[2] · Stéphane Mancini[4]

Received: 12 December 2019 / Accepted: 10 May 2021

Abstract In confronting the "Memory Wall", the design of embedded vision systems exhibits many challenges regarding design cost, energy consumption, and performance. This paper considers a variant of the Job Shop Scheduling Problem with tooling constraints, arising in this context, in which the completion time (makespan) is to be minimized. This objective corresponds to the performance of the produced circuit. We discuss different formulations using integer linear programming and point out their characteristics, namely the size and the quality of the linear programming relaxation bound. To solve this scheduling problem with large size, we compare various approaches, including a Constraint Programming model, two constructive greedy heuristics, two models of LocalSolver, a Simulated Annealing algorithm, and a Beam Search algorithm. Numerical experiments are conducted on 16 benchmark instances from the literature and 12 real-life non-linear image processing kernels for validating their efficiency.

Keywords Embedded vision systems · Scheduling · Makespan · Integer Linear Programming · Constraint Programming · Greedy Algorithms · LocalSolver · Simulated Annealing · Beam Search

1 Introduction

Electronic devices are now widespread and more than ever their design requires efficient optimization algorithms to drastically reduce their cost, increase performance, and improve energy consumption. Among these devices, embedded

✉ Khadija Hadj Salem
[1] Université de Tours, LIFAT EA 6300, CNRS, ROOT ERL CNRS 7002, 37200 Tours, France
E-mail: khadija.hadj-salem@univ-tours.fr
[2] Univ. Grenoble Alpes, Grenoble INP, GSCOP, 38031 Grenoble, France
[3] Univ. Grenoble Alpes, Grenoble INP, LCIS, 26000 Valence, France
[4] Univ. Grenoble Alpes, Grenoble INP, TIMA, 38031 Grenoble, France

vision systems are one of the most demanding because they process a huge amount of data acquired by high-resolution imaging sensors. As an example, it is now common to have electronic devices incorporating neural networks for video processing. These devices heavily make use of stencil-like processing and an algorithm called "kernel" is iterated over a nest of loops to produce an array of data from one or several input arrays. Due to the very high amount of data, such kernels make things difficult for processors because of the penalty of memory accesses. Standard data caches are inefficient without optimization of both the scheduling of processing and the sequence of data accesses. At the opposite of GPUs, for which the programmer has to accommodate to a generic memory framework, this work focuses on memory management of hardwired kernels, for which a specifically tailored memory subsystem can be designed. Then it is possible to deeply optimize the data management with efficient off-line algorithms, by exploring different formulations, in relation to a model of the hardwired computing unit and its memory subsystem.

The optimization of array processing has a long history, from the seminal work of Feautrier [14] to more recent one [12]. Preserving the code functionality, these optimizations are related to compilation techniques and their goal is to re-organize both the sequence of computations and the cache updates in a way to improve both the time locality and the spatial locality of memory references. These methods are called "linear" because they assume that array references are linearly related to loop indices through an integer algebraic relationship, and the scheduling is such that the time point to produce an output is also linearly related to loop indices. Related scheduling techniques are efficient but these assumptions exclude many applications such as image scaling and rotation, homographies, and so on.

Indeed, many real-time video processing uses "non-linear kernels" whose access patterns are not linearly related to loop indices. These non-linear kernels are used to correct non-linear optical systems such as fish-eye lenses, ego-motion estimation, cylindrical or spherical projection of 3D video, convolution kernels, and many others (see [31] and [8]). To optimize the memory management of non-linear kernels, the proposed technique relies on paving the space of both loop indices and input data by regular tiles and managing their movement from a huge external memory and a buffer close to the computing unit. The goal is to benefit the already loaded data and reduce the amount of data loaded from external memory. However, due to the non-linear access pattern, the amount of required input data called the "footprint" is not constant over the tiles. In this setting, the optimization process is to find a sequence of computations such that computing an output tile may benefit from the data already loaded for the previous output tiles. Unlike linear methods, the challenge here is that there is no matrix or linear relationship to benefit from, and *Combinatorial Optimization* (CO) methods are required.

The remainder of this paper is structured as follows. In section 2, we give a brief description of the context of embedded vision systems and a clear explanation about the related optimization problems. In section 3, the optimization problem under consideration is formally described. Some of its complexity

analysis results and its main lower bounds, and an example to illustrate the problem are then given. In section 4, three different integer linear programming models, as well as some dominance properties to speed up the search for an optimal solution, are presented. Sections 5—10 are devoted to solution procedures, including Constraint Programming, two greedy heuristics, two models of LocalSolver, a Simulated Annealing, and a Beam Search. Section 11 gives a detailed description and analysis of the computational results obtained by running the proposed approaches on benchmark instances and discusses each approach's performance. Finally, the paper concludes with a discussion on future research directions in Section 12.

2 Background and motivation

To address the challenge previously introduced, one co-designed architectural solution was proposed by Mancini and Rousseau [25]. Their solution, called `Memory Management Optimization` (MMOpt), creates an ad-hoc memory hierarchy suited for non-linear kernels. MMOpt takes as input a non-linear kernel, such as the one shown in Fig. 1, analyzes its access patterns, and computes a schedule of both the computations and the data movement between the external memory and internal buffers. It finally outputs a configuration of the so-called TPU (Tile Processing Unit), together with the information needed to orchestrate its operational behavior. The basis of this optimization is to tile both the iteration space of the kernel and the input and output data structures.

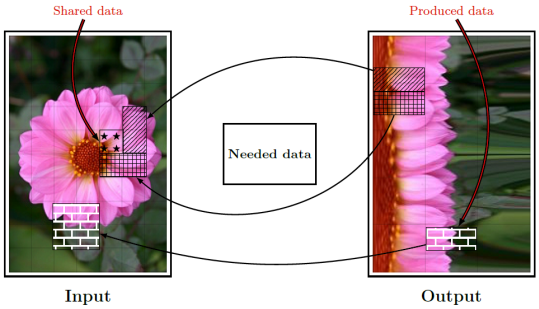


Fig. 1: The disparity of non-linear kernels, namely a polar transform in this picture, makes standard optimizations inefficient.

As shown in Fig. 2, the TPU is made of a `Prefetching Unit` that loads data from external memory to local buffers, and a `Processing Engine` (PE), that computes the output data from the input data contained in the buffers. This architecture allows continuous computations: prefetches being carried out in parallel with the computations. For this scheme to work, prefetches have to be determined in advance. In fact, in MMOpt, both prefetches and computations

are orchestrated according to a fixed schedule generated and integrated into the TPU.

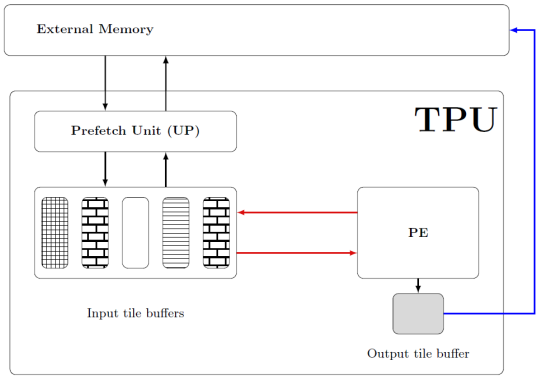


Fig. 2: Architecture template of the TPU

TPUs produced by MMOpt embed schedules for the prefetches of input tiles and output tiles computations (see Fig. 3). In this figure, output tile computing and input tile prefetching are scheduled simultaneously. It is also possible to have pauses in between computations, to limit the number of necessary buffers. A buffer can store any tile, but only one at a time. The optimized schedule will impact the three design characteristics of the generated computing unit in the following way: the number of buffers of the TPU will account for most of its area; the number of prefetches reflects the main part of the energy consumption[1]; and the performance is related to the total completion time to compute all the tiles of an image.

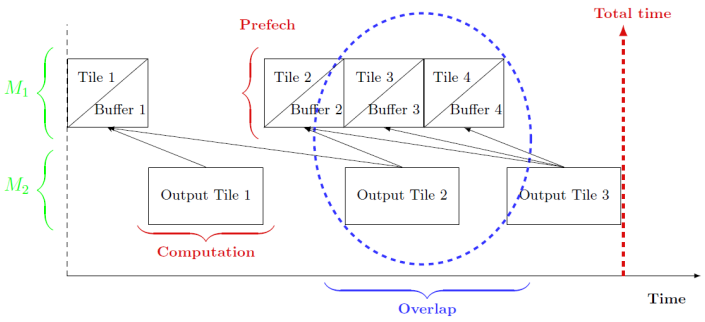


Fig. 3: Prefetches and computations schedules

[1] In the field of computer design, memory transfers are known to be a major part of energy consumption, up to 80% of the total energy

Following this optimization challenge, we state it as a concrete multi-objective optimization problem, called `3-objective Process Scheduling and Data Prefetching Problem` (3-PSDPP) (see [19]) with two objectives being parameters of the schedules themselves — the number of prefetches and the total makespan — and one parameter is the number of buffers of the TPU. They correspond to the energy consumption, respectively performance, and size/cost of the circuit. Since the use of combinatorial methods for optimizing the running of the TPU produced by the MMOpt tool is still an emerging field, we found only one systematic study of the published literature of MMOpt from 2012, done by Mancini and Rousseau [25]. This study is the only generic proposition that allows a significant performance improvement and can be used for all non-linear kernels.

To the best of our knowledge, the 3-PSDPP scheduling problem has not been studied before in the *Operational research* (OR) literature. In contrast, since 2014, this problem presents the basic topic of [19]. This electronic problem is formalized as a 3-objective scheduling problem with clearly delineated inputs and outputs in this study. A set of several constructive greedy heuristics, aiming at solving benchmarks from real-life non-linear image processing kernels, were developed. A more detailed description of the proposed model together with a list of all these algorithms can be found in [18] and [19].

This paper addresses one of the three natural single-objective sub-problems of 3-PSDDP, called `Minimum Completion Time of 3-PSDDP` (MCT-PSDPP), in which the makespan is to be minimized.

3 Minimum Completion Time of 3-PSDPP: MCT-PSDPP

3.1 Assumptions

Each TPU produced by MMOpt from [25] has to satisfy the following assumptions:

1. The input tile sizes are identical, and each input tile fits exactly into one buffer.
2. There is no distinction between buffers, i.e., any input tile may be prefetched into any buffer.
3. All input (respectively output) tiles, as well as the subset of input tiles required to compute each output tile, are known in advance.
4. Only one input (respectively output) tile can be prefetched (respectively computed) at a time.
5. The prefetch operations and the computation steps may be carried out simultaneously.
6. Input tile prefetch (respectively output tile computation) times are constant and identical.

3.2 Problem Description and Notation Definition

Formally, the scheduling problem MCT-PSDPP can be described as follows. Let $\mathcal{Y} = \{1, \ldots, Y\}$ be a set of Y independent non-preemptive output tiles (also called tasks) to be computed, and let $\mathcal{X} = \{1, \ldots, X\}$ be the set of X input tiles to be prefetched from the external memory to the internal buffers. For each output tile $y \in \mathcal{Y}$, we denote by $\mathcal{R}_y$ the subset of input tiles required for its computation (called prerequisites). These prerequisites tiles have to be prefetched from the external memory and must be present in the buffers during the corresponding computation step. Likewise, for each input tile $x \in \mathcal{X}$, let $\mathcal{R}_x$ be the subset of output tiles for which x is a prerequisite.

We assume that the number of buffers is unlimited, which means that each prefetch is performed in a dedicated buffer. Also, the processing time of a prefetch step and a computation step, respectively α and β, are input parameters.

The underlying problem is to determine:

(i) the schedule of computations $(c_j)_{j\in\mathcal{M}}, \mathcal{M} = \{1, \ldots, Y\}$, where for each computation step j, $c_j = (s_j, u_j)$ encodes the assignment of the output tile s_j to the computation starting time u_j.
(ii) a corresponding schedule of prefetches $(p_i)_{i\in\mathcal{N}}, \mathcal{N} = \{1, \ldots, X\}$, where $p_i = (d_i, b_i, t_i)$ encodes for each prefetch step i which input tile d_i is prefetched in which buffer b_i and at the prefetch starting time t_j.

The objective is to minimize the makespan, denoted by C_{max}, which means the total time it takes for the whole operation of the TPU from the beginning of the first prefetch to the end of the last computation of one full image.

To analyze the complexity of the MCT-PSDPP, several trivial variants, which can be solved in polynomial time, can be studied. For example, we first consider the case when the $\alpha > \beta Y$, in which the optimal makespan C^*_{max} is given by the formula $\alpha X + \beta \min_{x\in\mathcal{X}} |\mathcal{R}_x|$. Similarly, we consider the case when $\beta > \alpha \max_{y\in\mathcal{Y}} |\mathcal{R}_y|$, in which the optimal makespan C^*_{max} is given by the formula $\alpha \min_{y\in\mathcal{Y}} |\mathcal{R}_y| + \beta Y$. In the case when α equals to β and the cardinal of the set $\mathcal{R}_y, \forall y \in \mathcal{Y}$ does not exceed two required input tiles per each output one, the MCT-PSDPP is a trivial problem that belongs to the class $\mathcal{P}$. However, we have proved that a particular case of MCT-PSDPP, when $\beta = \alpha = 1$, is $\mathcal{NP}$-Hard, by giving a polynomial reduction from the "k-weak visit problem described in [6]. A detailed description of the proof can be found in [24].

In the literature (see Hadj Salem et al. [19]), there exist some lower bounds on the makespan $C_{\max}$ for the MCT-PSDPP, which are:

$$lb_1 = \alpha|\mathcal{X}'| + \beta \tag{1}$$

$$lb_2 = \alpha \min_{y\in\mathcal{Y}} |\mathcal{R}_y| + \beta Y \tag{2}$$

$$lb_3 = \alpha \min_{y\in\mathcal{Y}} |\mathcal{R}_y| + \beta Y' \tag{3}$$

Note that the $\mathcal{X}'$ denotes the set of input tiles that are required at least once for the computation of an output tile. In the same way, note that the Y' is the number of output tiles that are computed immediately after the last prefetch step in the prefetches/computations schedule. Thus, the makespane C_{max} is lower bounded by the maximum between the three lower bounds lb_1, lb_2 and lb_3: $lb_{C_{max}} = \max\{lb_1, lb_2, lb_3\}$.

3.3 Formulation as a non-classical scheduling problem

Extending the well-known three fields $\alpha|\beta|\gamma$ classification scheme for the scheduling problems, suggested by Graham et al. [17] (see also Lawler et al. [22]),—where α defines the machine environment, β defines the characteristics of the jobs, and γ defines the objective function that is to be minimized (max or min)—the MCT-PSDPP can be considered as an extension of some of the classical scheduling problems.

Firstly, if there is no shared prerequisites, this particular case of MCT-PSDPP can be seen as a Flow-shop Scheduling Problem (FSP), denoted as $F2|p_{1j} = \alpha|(\mathcal{R}_y)_{y\in\mathcal{Y}}|; p_{2j} = \beta|C_{max}$ (see Garey et al. [15] and Pinedo [27]). In this case, the problem is solvable in polynomial time by the well-known Johnson's algorithm (see Garey et al. [15]). A more detailed description of this variant of the MCT-PSDPP, as well as the adaptation of Johnson's algorithm to solve it, can be found in [24].

Secondly, the MCT-PSDPP can also be seen as a single machine scheduling problem with tool changes "Tool Switching Problem" (ToSP), also called *job Sequencing and tool Switching Problem* (SSP), where the objective function is to minimize the makespan.
The ToSP involves optimally sequencing jobs and assigning tools to a capacitated magazine to minimize the number of tool switches. It arises from computer and manufacturing systems, and it has been proved by Crama et al. [13]) as a $\mathcal{NP}$-complete combinatorial optimization problem. Different exact and heuristic methods have been defined to deal with this problem (see Tang et Denardo [30]; Bard [7]; Privault et Finke [28]; Laporte et al. [21]; Amaya et al. [5] and Catanzaro et al. [11]). A comprehensive review of the literature that summaries the current research results on the ToSP is provided by Calmels in [10].

By comparing our MCTP-PSDPP to the uniform variant of ToSP, we can state that both input and output tiles $(\mathcal{X}, \mathcal{Y})$ are regarded as ToSP data (tools, jobs). The incidence matrix Tools×Jobs can then be regarded as the requirements of input tiles needed to compute all the output tiles $(\mathcal{R}_y)_{y\in\mathcal{Y}}$. However, the number of buffers, which is analogous to the tool magazine's capacity, is unlimited.
Besides, MCTP-PSDPP involves determining a computation sequence and its corresponding prefetch (two independent sequences). Both output tile computing and input tile prefetching are scheduled simultaneously, and the makespan C_{max} is minimized. Notice that, compared to the ToSP, the MCT-PSDPP does not

have buffer limits; this changes the problem structure and makes dominance properties much more useful (we describe these dominance properties in Subsection 4.3).

In summary, this analysis can be considered as an interesting theoretical study to relate our MCT-PSDPP to similar scheduling problems known in the OR literature. This study led us to easily adapt some methods to solve our problem and some of its variants.

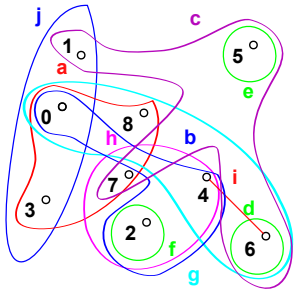
3.4 Illustrative Example

To illustrate the studied problem, MCT-PSDPP, we present the following example. Consider the input data given in (Tang et Denardo [30]) for the case where:

- $Y = 10$ output tiles ($\mathcal{Y} = \{a, b, c, d, e, f, g, h, i, j\}$);
- $X = 9$ input tiles ($\mathcal{Y} = \{0, \ldots, X-1\}$);
- $\mathcal{R}_{\mathcal{Y}} = \big[\{0,3,7,8\}, \{0,2,4\}, \{1,5,6,7\}, \{6\}, \{5\}, \{2\}, \{0,4,6,8\}, \{2,4,7\}, \{4,6\}, \{0,1,3\}\big]$ (see Fig.4);
- $\alpha = \beta = 1$ unit of time (for the simplicity of the problem).

$$\mathcal{R}_{\mathcal{XY}} = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{array} \begin{array}{c} \begin{array}{cccccccccc} a & b & c & d & e & f & g & h & i & j \end{array} \\ \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

(a) Incidence matrix $\mathcal{R}_{\mathcal{XY}}$



(b) Hypergraph $\mathcal{H}_{\mathcal{XY}}$

Fig. 4: An example of an instance of the MCT-PSDPP (Tang et Denardo, 1988) [30]: an incidence matrix $\mathcal{R}_{\mathcal{XY}}$ and its corresponding Hypergraph $\mathcal{H}_{\mathcal{XY}}$

Note that each instance of MCT-PSDPP can be also represented as a *bipartite graph*[2], denoted $B_{\mathcal{XY}}$, where $\mathcal{U} = \mathcal{X}$ (set of input tiles) and $\mathcal{V} = \mathcal{Y}$ (set of output tiles).

A feasible solution ϕ for MCT-PSDPP is given in Fig. 5, where $C_{max} = 14$ units of time.

In this schedule, the tile b is computed after prefetching all its required tiles $(0, 2, 4)$. The tile f is immediately computed because it does not need a

[2] **Bipartite graph** $B = (\mathcal{U}, \mathcal{V})$: consists of a set of vertices $\mathcal{U}$, a disjoint set of vertices $\mathcal{V}$, and a set of edges $\mathcal{E} \subset \mathcal{U} \times \mathcal{V}$.

Fig. 5: A feasible solution with $C_{max} = 14$ units of time

new input tile (tile 2 is already prefetched). For computing the tile h in the third step, we prefetch only the tile 7 while reusing tiles 2 and 4, which were previously prefetched for computing tile b, and so on.

4 Mathematical Programming Models

Mathematical programming formulation is a natural way to tackle scheduling problems. In this section, three Integer Linear Programming (ILP) models are provided for solving the proposed problem MCT-PSDPP.

4.1 Position-based ILP model

Denote $\mathcal{M} = \{1, \ldots, Y\}$ as the set of Y positions in the computation sequence to be determined and let $\mathcal{N} = \{1, \ldots, X\}$ be the set of X positions in the prefetch sequence to be defined. We then define two sets of variables $\{c_{yj} | y \in \mathcal{Y},\ j \in \mathcal{M}\}$ and $\{p_{xi} | x \in \mathcal{X},\ i \in \mathcal{N}\}$ to model the problem under consideration. c_{yj} is a binary variable equal to 1 if output tile y is computed at position j and 0 otherwise. p_{xi} is also a binary variable equal to 1 if input tile x is loaded at prefetch position i and 0 otherwise. Moreover, let $u_j,\ j \in \mathcal{M}$ and $t_i,\ i \in \mathcal{N}$ (where $u_j,\ t_i \in \mathbb{N}^*$) be the start time of the jth computation step and the ith prefetch step, respectively. We also use variable C_{max} to define the makespan, where $C_{max} \in \mathbb{R}$. Finally, denote Λ, where $\Lambda = \alpha X + \beta Y$ as an upper bound on the makespan C_{max} for the MCT-PSDPP (used as a Big-M constraints).

We now present our *position-based ILP model*, here-after denoted as `MCT-1`, as follows:

$$\min\ C_{max}$$

Subject to

$$\sum_{j \in \mathcal{M}} c_{yj} = 1 \quad \forall\, y \in \mathcal{Y} \tag{4}$$

$$\sum_{y \in \mathcal{Y}} c_{yj} = 1 \quad \forall\, j \in \mathcal{M} \tag{5}$$

$$\sum_{i \in \mathcal{N}} p_{xi} = 1 \quad \forall\, x \in \mathcal{X} \tag{6}$$

$$\sum_{x \in \mathcal{X}} p_{xi} = 1 \quad \forall\, i \in \mathcal{N} \tag{7}$$

$$u_j - t_i \geq \alpha - \Lambda(2 - c_{yj} - p_{xi}) \forall\, y \in \mathcal{Y},\ j \in \mathcal{M},\ x \in \mathcal{R}_y, i \in \mathcal{N} \tag{8}$$

$$u_{j-1} + \beta \leq u_j \quad \forall\, j \in \mathcal{M} \backslash \{1\} \tag{9}$$

$$t_{i-1} + \alpha \leq t_i \quad \forall\, i \in \mathcal{N} \backslash \{1\} \tag{10}$$

$$C_{max} \geq u_Y + \beta \tag{11}$$

$$c_{yj} \in \{0, 1\} \quad \forall\, y \in \mathcal{Y},\ j \in \mathcal{M} \tag{12}$$

$$p_{xi} \in \{0, 1\} \quad \forall\, x \in \mathcal{X},\ i \in \mathcal{N} \tag{13}$$

$$u_j, t_i \geq 1 \quad \forall\, j \in \mathcal{M},\ i \in \mathcal{N} \tag{14}$$

The objective function represents the makespan C_{max}, i.e., the end time of the last computation step defined by $u_Y + \beta$: inequality (11), which is to be minimized. Equalities (4)—(7) are a set of assignment constraints, in which (4) satisfies the requirement that there is a unique output tile be assigned to each computation step (position j), while (5) ensures that each output tile must be computed in a unique position. In the same way, constraints (6) satisfy the requirement that there is a unique input tile be assigned to each prefetch step (position i), while (7) ensures that each input tile must be loaded in a unique position. Constraints (8) ensure that each output tile is computed according to the requirement, which means that if tile x is prefetched at step i ($p_{xi} = 1$) and required by the output tile y which is computed at step j ($c_{yj} = 1$), then this x must be present in the internal buffer during this computation. This also means that the start date of this computation u_j must be greater than or equal to the date of existence of the tile x ($t_i + \alpha$). Constraints (9) guarantee that the computation step j only begins when the computation step $j - 1$ is finished. Similarly, constraints (10) ensure that the prefetch step i only begins when the prefetch step $i - 1$ is finished. Finally, constraints (12)—(14) set the variables' domains.

4.2 Time-based ILP models

The idea of using time-indexed variables is originally proposed here. We then give two ILP models for tackling the problem. Consider a set $\mathcal{T} = \{1, \ldots, T\}$, where $T \in \mathbb{N}^*$, as the time interval needed for performing all prefetch and

computation steps. In this paper, we fix $T = \alpha \sum_{y \in \mathcal{Y}} |\mathcal{R}_y| + \beta Y$, which can be considered as an upper bound on the makespan C_{max}. Denote $\mathcal{K} = \{1, \ldots, \alpha - 1\}$ as the time interval at which an input tile can be loaded and let $\mathcal{S} = \{1, \ldots, \alpha\}$ be the time interval at which a prefetch step of an input tile was performed (the prefetch is complete and the input tile is present in the buffer). Similarly, let $\mathcal{L} = \{1, \ldots, \beta - 1\}$ be the time interval at which an output tile can be computed.

4.2.1 Process-Date-Indexed ILP Model:

We define three sets of new variables $\{c_{yt} | y \in \mathcal{Y},\ t \in \mathcal{T}\}$, $\{p_{xt} | x \in \mathcal{X},\ t \in \mathcal{T}\}$ and $\{e_{xt} | x \in \mathcal{X},\ t \in \mathcal{T}\}$, where:

c_{yt} : $\begin{cases} 1 \text{ if output tile } y \text{ is in computation process at time } t, \forall y \in \mathcal{Y}, \forall t \in \mathcal{T} \\ 0 \text{ otherwise} \end{cases}$

p_{xt} : $\begin{cases} 1 \text{ if input tile } x \text{ is in prefetch process at time } t, \forall x \in \mathcal{X}, \forall t \in \mathcal{T} \\ 0 \text{ otherwise} \end{cases}$

e_{xt} : $\begin{cases} 1 \text{ if input tile } x \text{ exists in buffer at time } t, \forall x \in \mathcal{X}, \forall t \in \mathcal{T} \\ 0 \text{ otherwise} \end{cases}$

Let C_{max} be the makespan, where $C_{max} \in \mathbb{R}$. The *Process-Date-Indexed ILP model*, here-after denoted as `MCT-2`, can be written as follows:

$$\min\ C_{max}$$

Subject to

$$\sum_{t \in \mathcal{T}} c_{yt} = \beta \quad \forall\ y \in \mathcal{Y} \tag{15}$$

$$\sum_{y \in \mathcal{Y}} c_{yt} \leq 1 \quad \forall\ t \in \mathcal{T} \tag{16}$$

$$c_{yt} - c_{yt-1} \leq c_{yt+l} \quad \forall\ y \in \mathcal{Y},\ t \in \{2, \ldots, T - \beta\},\ l \in \mathcal{L} \tag{17}$$

$$\sum_{t \in \mathcal{T}} p_{xt} = \alpha \quad \forall\ x \in \mathcal{X} \tag{18}$$

$$\sum_{x \in \mathcal{X}} p_{xt} \leq 1 \quad \forall\ t \in \mathcal{T} \tag{19}$$

$$p_{xt} - p_{xt-1} \leq p_{xt+k} \quad \forall\ x \in \mathcal{X},\ t \in \{\alpha, \ldots, T - \beta\},\ k \in \mathcal{K} \tag{20}$$

$$e_{xt} - e_{xt-1} \leq p_{xt-s} \quad \forall\ x \in \mathcal{X},\ t \in \{\alpha + 1, \ldots, T\},\ s \in \mathcal{S} \tag{21}$$

$$c_{yt} \leq e_{xt} \quad \forall\ y \in \mathcal{Y},\ t \in \mathcal{T},\ x \in \mathcal{R}_y \tag{22}$$

$$e_{xt} = 0 \quad \forall\ x \in \mathcal{X},\ t \in \mathcal{S} \tag{23}$$

$$C_{max} \geq t c_{yt} \quad \forall\ y \in \mathcal{Y},\ t \in \mathcal{T} \tag{24}$$

$$c_{yt} \in \{0, 1\} \quad \forall\ y \in \mathcal{Y},\ t \in \mathcal{T} \tag{25}$$

$$p_{xt}, e_{xt} \in \{0, 1\} \quad \forall\ x \in \mathcal{X},\ t \in \mathcal{T} \tag{26}$$

The objective function minimizes the makespan C_{max}, where $C_{max} \geq t c_{yt}, \forall t \in \mathcal{T}$: constraint (24). Constraints (15)—(17) are assignment constraints

for computation steps. In the same way, constraints (18)—(20) are a set of assignment constraints for prefetch steps. Constraints (21) impose that a prefetch of tile x must be counted whenever x is present at instant t but is not present at instant $t-1$. In other words, $\forall x \in \mathcal{X}, t \in \{\alpha+1,\ldots,T\}$, $e_{xt} = 1$ and $e_{xt-1} = 0$ imply $p_{xt-s} = 1$ (the prefetch of the input tile x ends at instant $t-1$). Constraints (22) ensure that the computation of the output tile y starts at instant t, when all its required tiles $x, \forall x \in \mathcal{R}_y$ are present in the internal buffer before instant t. Constraints (23) are initialization constraints. Finally, constraints (25)—(26) set the variables' domains.

Remark 1 *The makespan C_{max} can be expressed by introducing the following decision variable:*

$$\delta_t : \begin{cases} 1 & \text{if the whole treatment (all computation and prefetch steps) is not yet complete at instant } t,\ \forall t \in \mathcal{T} \\ 0 & \text{otherwise (finished)} \end{cases}$$

In this case, we slightly modify the formulation `MCT-2`*, where the objective function will be defined by* $\min \sum_{t \in \mathcal{T}} \delta_t$ *and constraints (24) can be rewritten using the following two inequalities (27) and (28).*

$$\sum_{y \in \mathcal{Y}} c_{yt} \leq \delta_t \qquad \forall\ t \in \mathcal{T} \tag{27}$$

$$\delta_{t-1} \geq \delta_t \qquad \forall\ t \in \{2,\ldots,T\} \tag{28}$$

4.2.2 End Date-Indexed ILP Model: `MCT-3`

We now describe a new formulation using two sets of new variables $\{f_{yt} | y \in \mathcal{Y},\ t \in \mathcal{T}\}$, $\{q_{xt} | x \in \mathcal{X},\ t \in \mathcal{T}\}$ and both $\{e_{xt} | x \in \mathcal{X},\ t \in \mathcal{T}\}$ and $\{\delta_t | t \in \mathcal{T}\}$ from the previous formulation `MCT-2`, in which:

$$f_{yt} : \begin{cases} 1 \text{ if output tile } y \text{ finishes to be computed at instant } t,\ \forall y \in \mathcal{Y}, \forall t \in \mathcal{T} \\ 0 \text{ otherwise} \end{cases}$$

$$q_{xt} : \begin{cases} 1 \text{ if input tile } x \text{ finishes to be prefetched at instant } t,\ \forall x \in \mathcal{X}, \forall t \in \mathcal{T} \\ 0 \text{ otherwise} \end{cases}$$

The *End Date-Indexed ILP model*, here-after denoted as `MCT-3`, can be stated as follows:

$$\min \sum_{t \in \mathcal{T}} \delta_t$$

Subject to

$$\sum_{t\in\mathcal{T}} f_{yt} = 1 \quad \forall\, y \in \mathcal{Y} \tag{29}$$

$$\sum_{y\in\mathcal{Y}} \sum_{s=t-\beta+1}^{t} f_{ys} \leq 1 \quad \forall\, t \in \{\beta,\ldots,T\} \tag{30}$$

$$\sum_{t\in\mathcal{T}} q_{xt} = 1 \quad \forall\, x \in \mathcal{X} \tag{31}$$

$$\sum_{x\in\mathcal{X}} \sum_{s=t-\alpha+1}^{t} q_{xs} \leq 1 \quad \forall\, t \in \{\beta,\ldots,T\} \tag{32}$$

$$e_{xt} - e_{xt-1} \leq q_{xt-1} \quad \forall\, x \in \mathcal{X},\ t \in \{\alpha+1,\ldots,T\} \tag{33}$$

$$f_{yt} \leq e_{xs} \quad \forall\, y \in \mathcal{Y},\ t \in \{\beta,\ldots,T\}, x \in \mathcal{R}_y, s \in \{t-\beta+1,\ldots,t\} \tag{34}$$

$$e_{xt} = 0 \quad \forall\, x \in \mathcal{X},\ t \in \mathcal{S} \tag{35}$$

$$f_{yt} = 0 \quad \forall\, y \in \mathcal{Y},\ t \in \{1,\ldots,\beta+\alpha|\mathcal{R}_y|-1\} \tag{36}$$

$$\sum_{y\in\mathcal{Y}} f_{yt} \leq \delta_t \quad \forall\, t \in \mathcal{T} \tag{37}$$

$$\delta_{t-1} \geq \delta_t \quad \forall\, t \in \mathcal{T}\backslash\{1\} \tag{38}$$

$$f_{yt} \in \{0,1\} \quad \forall\, y \in \mathcal{Y},\ t \in \mathcal{T} \tag{39}$$

$$q_{xt}, e_{xt} \in \{0,1\} \quad \forall\, x \in \mathcal{X},\ t \in \mathcal{T} \tag{40}$$

$$\delta_t \in \{0,1\} \quad \forall\, t \in \mathcal{T} \tag{41}$$

The objective function minimizes the makespan defined by $\min \sum_{t\in\mathcal{T}} \delta_t$. Constraints (29) ensure that for each output tile y, there is an instant t in which this tile is computed, while constraints (30) guarantee that for each instant t, there is at most one prefetch of an input tile that will be finished at this time. In the same way, constraints (31) ensure that for each input tile x, there is an instant t in which this tile is loaded, while constraints (32) guarantee that for each instant t, there is at most one computation of an output tile that will be finished at this time. Constraints (33)—(35) imply constraints (21)—(23), respectively. Constraints (36) are initialization constraint for computation steps. Constraints (37)—(38) imply constraints (27)—(28), respectively. Finally, constraints (39)—(41) set the variables' domains.

Remark 2 *Constraints (34) can be rewritten as:*

$$\beta|\mathcal{R}_y| f_{yt} \leq \sum_{x\in\mathcal{R}_y} \sum_{s=t-\beta+1}^{t} e_{xs} \forall\ y \in \mathcal{Y},\ t \in \{\beta,\ldots,T\} \tag{42}$$

This constraint ensures that for each output tile y and each instant t, if the computation of y ends at t, then there is a time interval of time $\{t-\beta+1,\ldots,t\}$

to which all the prerequisites of y (given by $\mathcal{R}_y$) are prefetched. That is to say, the sum $\sum_{s=t-\beta+1}^{t} e_{xs}$ *must be greater than or equal to the time required for a computation step (β) multiplied by the number of prerequisites of y.*

For example, for any instance with $Y = 10$ output tiles to compute and $X = 9$ input tiles to load, the inequality (34) has 8795 constraints while (42) has 2174 constraints. We can then say that this inequality reduces the number of constraints that significantly impact solving performance.

4.3 Dominance Properties

In this subsection, we introduce some dominance properties which should apply for the `MCT-1` as well as for both `MCT-2` and `MCT-3`. These properties can be stated as "dominance rules" whose aim is to reduce the solution space of a problem (to reduce the searching scope) by adding new constraints to speed up the search process. In our case, we used these dominance properties as a preprocessing step that aims to reduce the search-space of the variables, or directly in building interesting solutions, or even a subset of solutions in which it is sufficient to search for optimal solutions.

Property 1 *(Tiles computation).*
Without loss of generality, we can assume that in an optimal solution $\mathcal{R}_{y_1} \subseteq \mathcal{R}_{y_2}, \forall (y_1, y_2) \in \mathcal{Y}$ implies that output tile y_1 must precede output tile y_2.

The property 1 can be simply described by the following inequalities: $\forall y_1, y_2 \in \mathcal{Y}$ *and* $\mathcal{R}_{y_1} \subseteq \mathcal{R}_{y_2}$:

$$u_j c_{y_1 j} \leq u_j c_{y_2 j+1} + \beta \qquad \text{(for the MCT-1)} \tag{43}$$

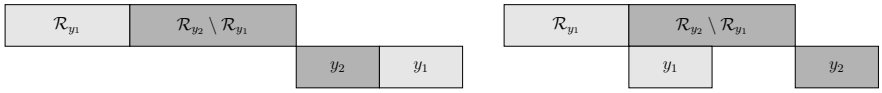
$$tc_{y_1 t} \leq tc_{y_2 t} + \beta \qquad \text{(for the MCT-2)} \tag{44}$$

$$tf_{y_1 t} \leq tf_{y_2 t} - \beta \qquad \text{(for the MCT-3)} \tag{45}$$

Proof 1 *(Property 1).*
Suppose there exists an optimal schedule where $\mathcal{R}_{y_1} \subseteq \mathcal{R}_{y_2}$ and y_2 precedes y_1 that contradicts dominance 1. *In this case, $C_{max} = \alpha|\mathcal{R}_{y_2}| + 2\beta$. Without breaking the precedence rule, we can exchange y_2 and y_1 in the schedule since both their prerequisites are loaded before s_1 and s_2. Doing so, we get $C_{max} = \alpha|\mathcal{R}_{y_1}| + \max(\alpha|\mathcal{R}_{y_2} \setminus \mathcal{R}_{y_1}|, \beta) + \beta$ which is at most $\alpha|\mathcal{R}_{y_2}| + 2\beta$. This contradicts the assumption of the existence of an optimal schedule where $\mathcal{R}_{y_1} \subseteq \mathcal{R}_{y_2}$ and y_2 precedes y_1 contradicting* dominance rule 1. *Thus there exists no optimal schedule that contradicts* dominance 1. *Figure. 6 shows an example when applying the dominance 1 allows getting a smaller C_{max}.*

□

Property 2 *(Tiles requirement).*
If all output tiles require an input tile, $\exists x \in \mathcal{X} / \forall y \in \mathcal{Y}, x \in \mathcal{R}_y$, then this input tile must be prefetched at the first prefetch step in an optimal schedule.



(a) y_2 before y_1 (b) y_1 before y_2

Fig. 6: Case where dominance 1 gets a smaller value

Proof 2 *(Property 2).*
Consider x_i the prerequisite which is required by all output tiles. It is a prerequisite of the first loaded input tiles. Since permuting prerequisites of the first output tiles do not change the completion time, there exists an optimal solution with the prerequisite required by all output tiles loaded first.

□

Property 3 *(Tiles utilization).*
If an output tile y requires all the input tiles, $\exists y \in \mathcal{Y}/\mathcal{R}_y = \mathcal{X}$; then there exists an optimal schedule in which this output tile is computed at the last computation step.

Proof 3 *(Property 3).*
Consider an output tile y which requires all the input tiles in $\mathcal{X}$. Let C_y be the completion time of y. The makespan C_{max} can be defined as $C_y + \beta k$, where $k \in \{0, \ldots, Y-1\}$ is the number of output tiles computed after y. As $C_y \geq \alpha X$, then $C_{max} \geq \alpha X + \beta k$. This means that minimizing C_{max} implies minimizing k. So, there exists an optimal schedule, where $k = 0$, which means that y is computed at the last computation step.

□

5 A Constraint Programming Approach

Constraint Programming (CP) is a declarative programming paradigm suitable to solve constraint satisfaction problems (CSPs). A CSP consists of a set of decision variables defined by a corresponding set of values (a finite domain) and a set of constraints that limit the possible combination of variable-value assignments. After a model of the problem is created, the solver interleaves two main steps: *constraint propagation*, where inconsistent values are removed from variables domains, and *a search procedure*. CP has been widely used to solve scheduling problems.

To present the `CP-1` model for the MCT-PSDPP, we use IBM ILOG optimization suite and the *docplex python module* [1] to program our model. We first define two set of variables as follows:

- I_1: the interval variable for each input tile (prerequisite) $x \in \mathcal{X}$;
- I_2: the interval variable for each output tile (task) $y \in \mathcal{Y}$;

The objective is to minimize the makespan, which is denoted by the following expression:

$$\max\{\texttt{end\_of}(I_2[y], \forall y \in \mathcal{Y})\ ^{3}\ \}$$

subject to the following constraints:

$$\texttt{no\_overlap}(\texttt{I}_1[\texttt{x}])\ ^{4} \quad \forall\ x \in \mathcal{X} \tag{46}$$

$$\texttt{no\_overlap}(\texttt{I}_2[\texttt{y}]) \quad \forall\ y \in \mathcal{Y} \tag{47}$$

$$\texttt{end\_before\_start}(\texttt{I}_1[\texttt{x}], \texttt{I}_2[\texttt{y}])\ ^{5} \quad \forall\ x \in \mathcal{X}, \forall\ y \in \mathcal{Y} \tag{48}$$

- Constraints (46) ensure that each input tile x is loaded at a prefetch instant only once.
- Constraints (47) ensure that each output tile y is computed at a computation instant only once.
- Constraints (48) mean that for each each x precedes y, x must finish before y starts. That is to say that an output tile y can be computed only after prefetching all its required tiles x.

6 Constructive Greedy Algorithms

The MCT-PSDPP was considered for the first time by Hadj Salem et al. [19]. They have developed a set of two constructive greedy algorithms, called as follows:

a **Earliest Computations for MCT** (ECM):
The ECM algorithm's main idea is to compute the output tiles at the earliest while satisfying the input tiles requirement constraint. Note that the prefetches are sequenced in their decreasing order of the number of occurrences $O_c(x), \forall x \in \mathcal{X}$.

b **Computation Grouping for MCT** (CGM):
The CGM algorithm's main idea is to find a set of groups $\mathcal{G}$ to define the computations sequence. A *Group* $\mathcal{G}$ defines a set of output tiles $y \in \mathcal{Y}$ which share the same required input tiles and will be successively computed after y. More formally, considering an output tile y, a *Group* $\mathcal{G}$ of y is defined by $\mathcal{G}(y) = \{g : g \in \mathcal{Y}, g \neq y,\ and\ \mathcal{R}_g \subseteq \mathcal{R}_y\}$. To determine this computations sequence, we first construct the set of groups $\mathcal{Y}'$, associated to the set output tiles $y \in \mathcal{Y}$, while ensuring that each output tile y belongs to exactly one group $\mathcal{G}$. Then, the computations are sequenced in their increasing order of $|\mathcal{R}_y|, \forall y \in \mathcal{Y}'$ and $\mathcal{Y}' \subseteq \mathcal{Y}$.

For all these algorithms, the number of prefetches N equals its lower bound $lb_N = |\mathcal{X}'|$ and the number of buffers Z equals its number of required input tiles $|\mathcal{X}'|$.
The flowchart in Fig. 7 summarizes the basics steps of each of these methods. However, a more detailed description of these algorithms and an illustrative example can be found in [19].

[3] `end_of()`: End of an interval variable.

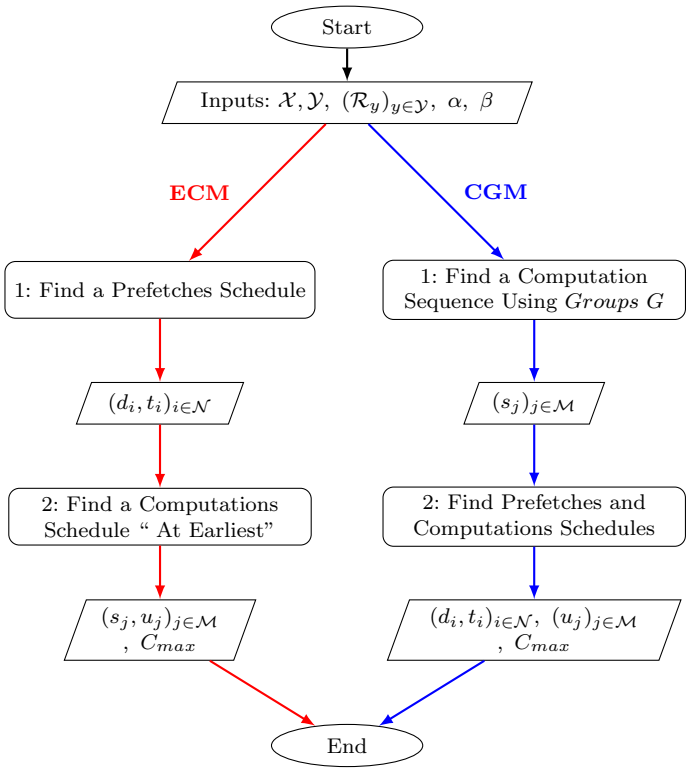


Fig. 7: Flowchart of ECM and CGM algorithms [19]

7 Split Algorithm

Before going into specific meta-heuristics, we present a `Split algorithm` for MCTP-PSDPP. Split algorithms are a way to search in a smaller search space. A good example of such an application, in the vehicle routing literature, can be found in [32], in which a split algorithm in O(n) is used to partition a solution (represented as a giant tour without occurrences of the depot) into separate routes with minimum cost. In our case, we apply it as follows: given a permutation on output tasks, there exists an optimal schedule of input tasks relative to the output permutation. First, it consists of scheduling (prefetching) all prerequisites (input tasks) of the first scheduled output task, then all prerequisites of the second scheduled output tasks, and so on until all input tasks are scheduled (prefetched). One can prove that it leads to an optimal input task schedule by a permutation argument. We use this routine in one LocalSolver model, the Beam Search (BS) and the Simulated Annealing (SA).

8 Solving using LocalSolver

LocalSolver (LS) is a local search-based mathematical programming software. More information about it can be found in [9].

We propose two LocalSolver models (`LS-standard` and `LS-split`). The first one is a naive approach using a straightforward model reflecting the `MCT-1` formulation. It is implemented as the MIP defined in this paper. We take advantage of the set modeling in LocalSolver 7.5. The second one uses the split routine defined before. It searches over permutation and defines a custom objective function that computes the optimal input task schedule, then evaluates the resulting solution.

9 The Proposed Simulated Annealing

Simulated Annealing (SA) is a well-known meta-heuristic that belongs to the class of randomized local search algorithms, which are known as threshold accepting algorithms. Standard SA has been widely used in optimization and present in most of the textbooks [16]. We use the split method defined before.

Like all others meta-heuristic methods, we need to define the different parameters as follows:

1. **Initial solution:** we generate randomly a permutation $(s_j)_{j\in\mathcal{Y}}$ of $|\mathcal{Y}|$ and compute the permutation $(d_i)_{i\in\mathcal{N}}$ of $|\mathcal{X}|$.
2. **Neighborhood**: the neighborhood is defined by all the possible swaps of two elements in the permutation $(s_j)_{j\in\mathcal{Y}}$. It consists of $O(|\mathcal{Y}|^2)$ elements.
3. **Evaluation function:** we measure the makespan C_{max}.
4. **Acceptance scheme:** We accept a non-improving solution (of difference Δ between the reference solution at iteration k) if a random number between 0 and 1 is less than $\exp(\frac{-\Delta}{\exp -k/10})$

Algorithm 1 shows the pseudo-code of our proposed *Simulated Annealing algorithm.*

10 Iterative beam search algorithm

Beam Search (BS) has been used successfully to solve many scheduling problems (see [26, 29]). Beam Search is a tree search algorithm that uses a beam size parameter (D). Beam Search behaves like a truncated *Breadth First Search (BFS)*. It only considers the best D nodes on a given level. The others are fathomed. Usually, we use the dual bound of a node to choose the most promising nodes. In our situation, MCTP-PSDPP is a minimization problem, we use as a guide the value of the lower bound defined by the following formula:

$$lb_{C_{max}} = \text{idle time on machine } M_2 + |\mathcal{Y}| \tag{49}$$

Algorithm 1: Simulated Annealing algorithm

Input : $\mathcal{X}, \mathcal{Y}, \mathcal{R}_y, \alpha, \beta$
Output: permutation of $\mathcal{Y}$: $(s_j)_{j\in\mathcal{M}}$
Result: optimal makespan C_{max}

```
1  while not stoppingCriterion do
2  |  Initialize temperature t_0 and s as a random solution for k ∈ {0,...K} do
3  |  |  Generate neighbour n ∈ ℕ(s) ; // n is a neighbour of s
4  |  |  C_max ← eval(n) − eval(s) ;
5  |  |  if C_max ≤ 0 then // We found a better solution
6  |  |  |  s ← n ;
7  |  |  else
8  |  |  |  s ← n with probability exp(−C_max/t_k) ;
9  |  |  end
10 |  end
11 end
```

It generalizes both a greedy algorithm (if $D = 1$) and a BFS (if $D = \infty$). An iterative scheme was recently proposed to solve various combinatorial optimization problems using a beam search or a variant of it (see [23]). It consists of successive runs of larger and larger beam search algorithms. This allows to get rapidly good solutions and, being able to improve them.

Algorithm 2 shows the pseudo-code of our proposed *Iterative beam search algorithm*. The algorithm runs multiple beam-searches starting with $D = 1$ (line 1) and increases the beam size (line 8) geometrically. Each run explores the tree with the given parameter D. At the end of the time limit, we report the best solution found so far (line 10). We start with an empty output permutation, then chose the first output task at the first level. Then the second output task on the second level. And so on.

Algorithm 2: Beam Search algorithm

Input : $\mathcal{X}, \mathcal{Y}, \mathcal{R}_y, \alpha, \beta$
Output: permutation of $\mathcal{Y}$: $(s_j)_{j\in\mathcal{M}}$
Result: optimal makespan C_{max}

```
1  D ← 1;
2  while time limit not exceeded do
3  |  Candidates ← root ; // root is an empty output permutation of 𝒴
4  |  while Candidates ≠ ∅ do
5  |  |  Children ← { children(n) | n ∈ Candidates } ; // children is a child
   |  |   node of the tree a limited set
6  |  |  Candidates ← best D of Children;
7  |  end
8  |  D ← D × 2;
9  end
10 Report best solution found;
```

11 Computational Experiments and Discussion

The experiments' goal is to evaluate the different proposed methods, including exact and meta-heuristics, for solving the MCT-PSDPP given two sets of benchmarks.

11.1 Parameter settings

All experiments were performed on an Intel Core i5 processor, 2.60 GHz machine, equipped with 4 GB of RAM and operating system Windows. We perform experiments with Gurobi Optimizer v7.5.1 and LocalSolver 7.5 using Python 3.6. The Constraint programming approach was implemented using IBM ILOG CP Optimizer. Simulated Annealing, as well as greedy algorithms, were implemented using Python version 3.6. Beam Search was implemented in C++. The CPU time limit for each run on each problem instance is 300 seconds. All our tests were carried out for the case where $\alpha = \beta = 1$ time unit. All the solvers are executed in their default settings with one thread unless specified otherwise.

11.2 Description of data-sets

Experiments were made using two kinds of data-sets possessing different characteristics.

Specifically we first considered a collection of 16 data-sets for the well-known ToSP, available in the literature (see [7], [20], [4], and [34]) and downloadable at [2].
Each data-set contains 5 random instances (i.e., incidence matrices or relations among input and output tiles) of the MCT-PSDPP (≡ ToSP), characterized by having the same number of output tiles (≡ jobs), input tiles (≡ tools). For the sake of simplicity, since instances sharing the same characteristics produce the same results on considered algorithms, we only present the first one among each class. As we can see from the Table 1, each data-set is also characterized by the vector of parameters $Y, X, X_{min}, X_{max}, Z$, where:

- $Y \in \{10, \ldots, 50\}$
- $X \in \{9, \ldots, 60\}$
- $X_{min} = \min_{y \in \mathcal{Y}} |\mathcal{R}_y|$
- $X_{max} = \max_{y \in \mathcal{Y}} |\mathcal{R}_y|$
- $Z \in \{4, \ldots, 30\}$. In our case, the Z is infinite number of buffers.

A specific instance with Y output tiles, X input tiles, and buffers' number Z is labeled as: $Z\zeta_Y^X$.
As described in [7], [20], [4], and [34], a generic instance in a given data-set is created by generating at random, for each output tile $y \in \mathcal{Y}$, the set $\mathcal{R}_y$ ($X_{min} \leq |\mathcal{R}_y| \leq X_{max}$), and with the restriction that no output tile is covered

by any other output tile in the sense that $\forall k, l \in \mathcal{Y}$ and $k \neq l : \mathcal{R}_k \nsubseteq \mathcal{R}_l$. In fact, data-sets belonging to the same group (e.g., datAx, datBx, and so on) differ from one on another by the number of input tiles X and the number of buffers Z.

Note that all our proposed algorithms perform in the same way for the different instances from the same group. So, in this paper, we present our numerical results (given, in the next sections, in both Tables 4 and 5) only on one of them.

A visualization of these instances can be found in [3]. It is a homemade web application helping to visualize hypergraphs, play with *ToSP* instances and solutions. Each row can be dragged and dropped at another position in the order by picking its pink square, similarly for columns and green squares.

Data-set	Y	X	X_{min}	X_{max}	Z	Label
datA1	10	9	2	4	4	$4\zeta_{10}^{9}$
datA2	-	10	2	4	4	$4\zeta_{10}^{10}$
datA3	-	15	3	6	6	$6\zeta_{10}^{15}$
datB1	15	12	3	6	6	$6\zeta_{15}^{12}$
datB2	-	20	3	6	6	$6\zeta_{15}^{20}$
datC1	20	15	3	8	8	$8\zeta_{20}^{15}$
datC2	-	16	3	8	8	$8\zeta_{20}^{16}$
datC3	-	20	4	10	10	$10\zeta_{20}^{20}$
datC4	-	30	9	24	24	$24\zeta_{20}^{30}$
datC5	-	36	9	24	24	$24\zeta_{20}^{36}$
datC6	-	40	11	30	30	$30\zeta_{20}^{40}$
datD1	30	25	4	10	10	$10\zeta_{30}^{25}$
datD2	-	40	6	15	15	$15\zeta_{30}^{40}$
datE1	40	30	6	15	15	$15\zeta_{40}^{30}$
datE2	-	60	7	20	20	$20\zeta_{40}^{60}$
datF	50	40	9	20	25	$25\zeta_{50}^{40}$

Table 1: Characteristics of the 16 data-sets of the ToSP

We then considered a set of 12 benchmarks from real-life non-linear image processing kernels already used by Mancini and Rousseau [25]. Note that the kernel's incidence matrices are our input, not the image processed by the kernel.

As we can see from Table 2, the benchmarks are variations of four kernels (fisheye, polar, fd resize, and fd haar) for which the input data structure (multi-resolution (an)isotropic mipmap input data) is modified. In fact, the first four kernels represent geometric non-linear transformations (see [31] and [8]). The last one, which represents a kernel of a face detection application based on Haar features, creates a pyramidal multi-resolution image (see [33]). The number of the input image tiles varies between 60 and 7000 input tiles. Similarly, the number of the output tiles varies between 60 and 3400 tiles, where Y_0 is

N°	Kernel	Input data type	Input tiles		Output tiles			Prerequisites	
			Dim	X	Dim	Y_0	Y	X_{min}	X_{max}
1	Test2D	image	2D	256	2D	64	64	4	4
2	Test2D PE	image	2D	64	2D	256	256	1	1
3	Fisheye	image	2D	176	2D	176	158	1	9
4	Fisheye	mipmap isotropic	3D	352	2D	176	158	2	13
5	Fisheye	mipmap anisotropic	4D	704	2D	176	158	3	21
6	Polar	image	2D	169	2D	112	112	2	8
7	Polar	mipmap isotropic	3D	845	2D	112	112	2	12
8	Polar	mipmap anisotropic	4D	4225	2D	112	112	5	20
9	Fd Resize	mipmap isotropic	3D	1280	3D	3520	1186	1	13
10	Fd Haar	pyramidal integral image	4D	7040	3D	2112	428	28	96
11	Cameleon	image	3D	1200	2D	1350	877	1	9
12	Cameleon Sd	image	3D	4800	2D	5400	3353	1	10

Table 2: Characteristics of the 12 benchmarks from real-life non-linear image processing kernels

the initial number of output tiles, and Y defines the number of tiles to be computed: $Y \leq Y_0$.

11.3 Experiments for ILP models

11.3.1 Comparison of the ILPs

We first define by R the average of $|\mathcal{R}_y|$: $R = \frac{1}{Y} \sum_{y \in \mathcal{Y}} |\mathcal{R}_y|$. Table 3 gives a comparison of the proposed ILP models: `MCT-1`, `MCT-2a`, `MCT-2b`, `MCT-3a` & `MCT-3b`, in terms of the number of variables and the number of constraints as well as the use of the Big-M constraints. These models are based on the formulations `MCT-1`, `MCT-2` and `MCT-3` (described in Section 4), where:

- `MCT-1`: is the *position-based ILP* model;
- `MCT-2a`: is the *Process-Date-Indexed ILP* model with a classical objective function of the makespan;
- `MCT-2b`: is the *Process-Date-Indexed ILP* model with an objective function as a decision variable defined by constraints (26) — (27);
- `MCT-3a`: is the *End-Date-Indexed ILP* model with the initial version of constraints (33);
- `MCT-3b`: is the *End-Date-Indexed ILP* model with the second version of constraints (33) defined by constraints (41).

As illustrated in Table 3, it is easy to see that `MCT-2a`, `MCT-2b`, `MCT-3a` & `MCT-3b` are equivalent in terms of the number of binary variables. They have the largest number of variables, but they also have some strong advantages. They do not contain the Big-M constraints that are known to weaken the linear relaxation and decrease ILP models' performance. Besides, using a binary variable to define the makespan C_{max} seems to be more favorable than the use of the classical ones.

Models	No. Variables		No. Constraints	Big-M
	binary	integer		
`MCT-1`	$X^2 + Y^2$	$X + Y + 1$	$\mathcal{O}(XY^2R)$	Yes
`MCT-2a`	$(2X + Y)T$	1	$\mathcal{O}((\alpha + \beta)Y^2R^2 + \alpha XY(\alpha R + \beta))$	No
`MCT-2b`	$(2X + Y)T + T$	0	$\mathcal{O}((\alpha + \beta)Y^2R^2 + \alpha XY(\alpha R + \beta))$	No
`MCT-3a`	$(2X + Y)T + T$	0	$\mathcal{O}(\beta Y^2R(\alpha R + \beta) + XY(\alpha R + \beta))$	No
`MCT-3b`	$(2X + Y)T + T$	0	$\mathcal{O}(Y^2(\alpha R + \beta) + XY(\alpha R + \beta))$	No

Table 3: Comparison of the ILP models: `MCT-1`, `MCT-2a`, `MCT-2b`, `MCT-3a` & `MCT-3b`

In contrast, `MCT-1` has the smallest number of variables (binary and integer). The most important disadvantage of `MCT-1` is that it uses a large number Λ in the constraint (8), known as the Big-M constraints. Because we can not find an efficient way to estimate Λ closely, an exact procedure to separate this constraint will be useless.
To compare the number of constraints for the different models, we need to study the size of R. In fact, if $R \leq \frac{X}{\alpha+\beta}$, both `MCT-2a` and `MCT-2b` are then better than `MCT-1` in terms of number of constraints. Furthermore, if $R \leq \frac{X}{\alpha\beta}$, `MCT-3a` has less constraints than `MCT-1`. Then, `MCT-3b` is always better.

11.3.2 Computational results

The first set of our computational results provides the gap (expressed in percentage) and CPU (expressed in seconds) values for the five ILP models, on the ToSP data-sets. These experiments were performed with Gurobi Optimizer, when enabling its proprietary cuts and presolve strategies.

Analysis of the gap: Fig. 8 shows a comparison of both `Gurobi gap` and `Gap` distribution using box-plots [6]. The `Gurobi gap` is defined as the difference between the best feasible solution and the best lower bound found by Gurobi at the end of the CPU time limit (300 s). However, the `Gap` is calculated using the following formula, where C^*_{max} is obtained by solving the CP model:

$$\frac{(C_{max} - C^*_{max})}{C^*_{max}} * 100 \tag{50}$$

Tables 4 and 5 give the detailed numerical results of the different versions of the ILP models for MCT-PSDPP using ToSP data-sets. For each of the five models, we give the following parameters:

– `lb`: the best lower bound found by Gurobi;

[6] *A box plot (or box-and-whisker plot) shows the distribution of quantitative data in a way that facilitates comparisons between variables or across levels of a categorical variable. Specifically, the bottom and the top of each box represent the first and third quartiles; the band inside the box represents the second quartile (the median); the ends of the whiskers represent the 9th percentile and the 91 percentile. Outliers are plotted as individual points.*

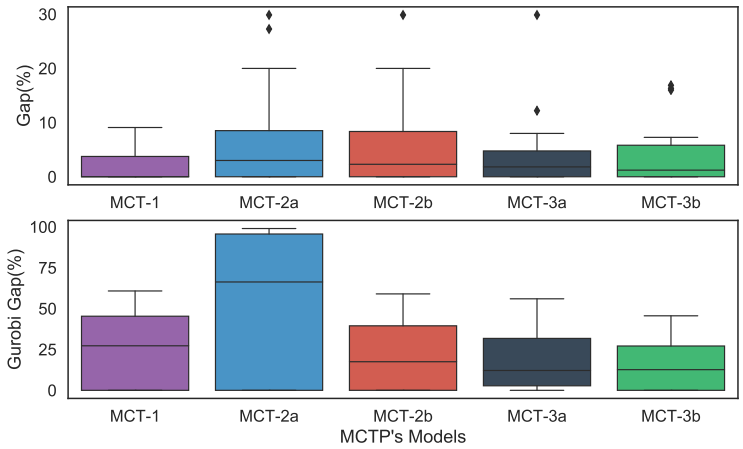


Fig. 8: Comparison of the gaps of models `MCT-1`, `MCT-2a`, `MCT-2b`, `MCT-3a` and `MCT-3b`

Id	C^*_{max}	MCT-1					MCT-2a					MCT-2b				
		lb	C_{max}	GGap	CPU	Gap	lb	C_{max}	GGap	CPU	Gap	lb	C_{max}	GGap	CPU	Gap
A1	12	12	**12**	**0**	**0.66**	**0**	12	**12**	**0**	**1.49**	**0**	12	**12**	**0**	**0.22**	**0**
A2	13	13	**13**	**0**	**4.62**	**0**	13	**13**	**0**	**0.89**	**0**	13	**13**	**0**	**0.27**	**0**
A3	16	16	**16**	**0**	**1.28**	**0**	16	**16**	**0**	**6.19**	**0**	16	**16**	**0**	**1.97**	**0**
B1	19	19	**19**	**0**	**5.86**	**0**	19	**19**	**0**	**12.41**	**0**	19	**19**	**0**	**1.20**	**0**
B2	22	22	**22**	**0**	**60.69**	**0**	22	22	**0**	**263.86**	**0**	19	22	13.6	300	**0**
C1	25	25	**25**	**0**	**58.54**	**0**	18	25	28	300	**0**	25	**25**	**0**	**216.12**	**0**
C2	27	20	27	25.9	300	**0**	15	27	44.4	300	**0**	25	27	7.4	300	**0**
C3	28	20	28	28.6	300	**0**	16	28	44.8	300	3.57	24	30	20	300	7.14
C4	41	20	41	51.2	300	**0**	3	42	92.9	300	2.43	28	43	34.9	300	4.87
C5	44	20	45	55.6	300	2.27	1	56	98.2	300	27.27	27	50	46	300	13.63
C6	49	20	51	60.8	300	4.08	2	51	96.1	300	4.08	29	50	42	300	2.04
D1	39	30	40	25	300	2.56	5	41	87.8	300	5.12	34	40	15	300	2.56
D2	50	30	53	43.4	300	6	3	52	94.2	300	4	34	56	39.3	300	12
E1	55	40	57	29.8	300	3.63	3	66	95.5	300	20	44	58	24.1	300	5.45
E2	77	40	84	25.4	300	9.09	1	100	99	300	29.87	41	100	59	300	29.87
F	75	50	80	37.5	300	6.66	2	89	97.8	300	18.66	54	90	40	300	20
Average		-	-	25.63	-	2.14	-	-	54.91	-	7.18	-	-	21.33	-	6.09

Table 4: Numerical results of ILP models for MCT-PSDPP: `MCT-1`, `MCT-2a` & `MCT-2b`

- C_{max}: the best feasible solution found by Gurobi Optimizer, when enabling its proprietary cuts and presolving strategies;
- `GGap`: the Gurobi Gap expressed in percentage (%);
- `CPU(s)`: the execution time expressed in seconds;
- `Gap`: the optimal gap, expressed in percentage (%).

In these tables, we highlight the optimally solved instances within the time limit (less than or equal to one CPU minute) with bold type. This means that `GGap` and/or `Gap` were equal to 0%.

Id	C^*_{max}	MCT-3a					MCT-3b				
		lb	C_{max}	GGap	CPU	Gap	lb	C_{max}	GGap	CPU	Gap
A1	12	12	**12**	**0**	**0.13**	**0**	12	**12**	**0**	**0.22**	**0**
A2	13	13	**13**	**0**	**0.30**	**0**	13	**13**	**0**	**0.27**	**0**
A3	16	16	**16**	**0**	**3.30**	**0**	16	**16**	**0**	**3.64**	**0**
B1	19	19	**19**	**0**	**1.43**	**0**	19	**19**	**0**	**2.92**	**0**
B2	22	19	22	13.6	300	**0**	21	22	4.5	300	**0**
C1	25	25	26	3.8	300	4	25	**25**	**0**	**11.27**	**0**
C2	27	26	27	3.7	300	**0**	27	**27**	**0**	**170.29**	**0**
C3	28	25	28	10.7	300	**0**	25	28	10.7	300	**0**
C4	41	31	46	32.6	300	12.19	33	42	21.4	300	2.43
C5	44	31	47	34	300	6.81	31	46	32.6	300	4.54
C6	49	34	51	33.3	300	4.08	33	57	42.1	300	16.32
D1	39	35	39	10.3	300	**0**	35	41	14.6	300	5.12
D2	50	37	54	31.5	300	8	37	58	36.2	300	16
E1	55	46	57	19.3	300	3.63	47	59	20.3	300	7.27
E2	77	44	100	56	300	29.87	49	90	45.6	300	16.88
F	75	59	78	24.4	300	4	59	79	25.3	300	5.33
Average		-	-	17.07	-	4.53	-	-	15.83	-	4.62

Table 5: Numerical results of ILP models for MCT-PSDPP: MCT-3a & MCT-3b

From the results shown in the Fig. 9 and in both Tables 4 and 5, we can make the following observations: (i) The four smaller instances (instances goes from size 9×10 to 15×12) are easily solved by each of the proposed ILP models. In contrast, none of the proposed ILP models is able to solve the nine bigger instances with up to 20 input/output tiles. The quality difference between the proposed ILP models is really visible with middle size instances (instance B2, C1, and C2), which have been solved only by some models. (ii) the best Gap results are with the MCT-1 model (average Gap: 2.14 %). However, both MCT-3a and MCT-3b models (average Gap, respectively: 4.53 % and 4.62 %) are not so far from MCT-1 model. (iii) The number of optimally solved instances is a good quality indicator for ILP models. The proposed ILP models that solve the most number of instances are MCT-1 and MCT-3b. (iv) The gurobi gap GGap provide a value of the solving state. MCT-3b model gives the best GGap value on our instance sets.

Fig. 9 shows a comparison of the last lower bound provided by Gurobi for each of the five ILP models in terms of line plots. The results show that MCT-3b is the tightest formulation for the MCT-PSDPP and that even if MCT-1 model provides the best average Gap, it gives a weaker lower bounds than the other three models (MCT-2b, MCT-3a and MCT-3b). The strength of the provided lower bound is a major factor to prove optimality.

Analysis of the solution times: Fig. 10 shows a comparison of the solution times of each of the five ILP models in terms of line plots.

As a general trend, we observed that both MCT-1 and MCT-2a are the slowest to compute the solved instances. Furthermore, the three ones MCT-2b, MCT-3a

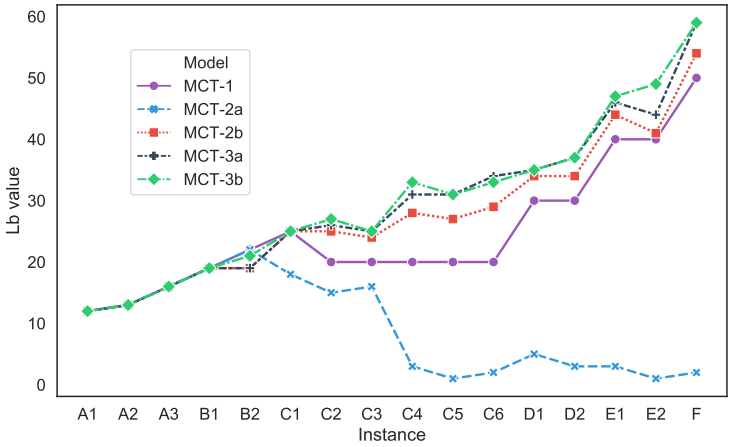


Fig. 9: Comparison of the lower bound of models `MCT-1`, `MCT-2a`, `MCT-2b`, `MCT-3a` and `MCT-3b` on the ToSP data-sets (Y are in either increasing)

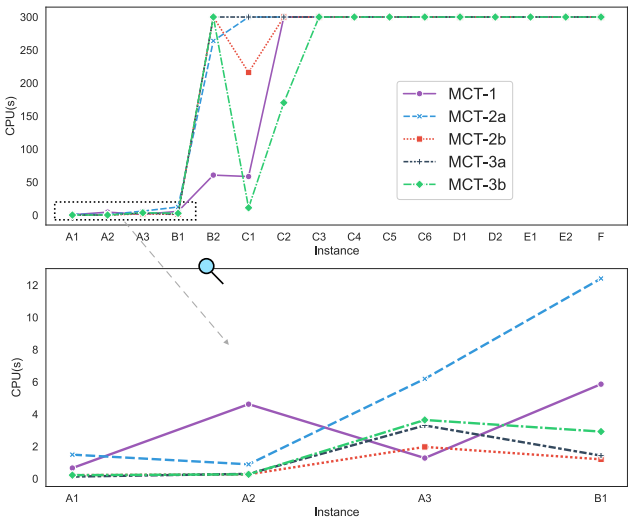


Fig. 10: Comparison of the solution times (expressed in seconds) of models , `MCT-2a`, `MCT-2b`, `MCT-3a` and `MCT-3b` on the ToSP data-sets (Y are in either increasing), where the second plot is a zoom-in of the dotted rectangle

and `MCT-3b` have a similar behavior. This could be explained by the constraint complexity analysis done before.

11.4 Experiments for Heuristics and Meta-heuristic methods

11.4.1 Computational results

The second set of results compares the makespan C_{max} obtained by each of our algorithms (Constraint Programming `CP`, Greedy algorithms `ECM` & `CGM`, LocalSolver `LS-standard` & `LS-split`, Simulated Annealing `SA` and Beam Search `BS`) and for each problem instance (described in section 11.2). The results are given in Table 7 (on the ToSP data-sets) and Table 8 (on the MMOpt kernels).

In theses tables, the first column `Id` refers to data sets, the second column indicates the lower bound value `lb` on the makespan C_{max} defined by the maximum of lb_1 and lb_2 given by the two Equations 1 and 2 and the third column shows the optimal makespan C^*_{max} obtained by the CP model. We also noted that both `lb` and C^*_{max} are the same value in the case of MMOpt kernels (see Table 5). Besides, they give, for each algorithm, the following parameters:

- `C`: the makespan C_{max} value;
- `G(%)`: the gap, expressed in percentage, calculated using the equality 50.

The last row in both Table 7 and 8 provides the average gains of the `G(%)` parameter for all the kernels.
In the same way, Table 6 gives the average solution times (expressed in seconds) taken by the different algorithms described in previous sections to solve ToSP instances and MMOpt kernels.

`Algorithm/Data-sets`	ToSP instances	MMopt instances
`CP`	2.33	22.33
`ECM`	$\ll 1$	≤ 1
`CGM`	$\ll 1$	42.33
`LS-standard`	≤ 60	14.91
`LS-split`	3.62	1.08
`SA`	≤ 1	$\simeq 250$
`BS`	$\ll 1$	≤ 1

Table 6: Average CPU(s) of CP, heuristics, and meta-heuristics methods for both ToSP and MMOpt data-sets

11.4.2 Convergence Analysis

We study the convergence of LocalSolver (for both models `LS-standard` and `LS-split`), Simulated Annealing `SA` and Beam Search `BS`, that gives us an

Id	lb	C^*_{max}	CP		ECM		CGM		LS-standard		LS-split		SA		BS	
			C	G(%)	C	G(%)	C	G(%)	C	G(%)	C	G(%)	C	G(%)	C	G(%)
A1	12	12	12	0	13	8.33	13	8.33	15	20	12	0	12	0	12	0
A2	12	13	13	0	13	8.33	14	7.69	16	18.75	13	0	13	0	13	0
A3	16	16	16	0	18	12.5	18	12.5	20	20	16	0	16	0	16	0
B1	19	19	19	0	22	15.79	20	5.26	22	13.6	19	0	19	0	19	0
B2	21	22	22	0	25	13.63	27	22.72	26	15.4	22	0	22	0	22	0
C1	23	25	25	0	29	16	29	16	28	10.7	258	0	25	0	25	0
C2	23	27	27	0	30	11.11	30	11.11	31	12.9	27	0	27	0	27	0
C3	24	28	28	0	32	14.2	31	10.71	32	12.5	28	0	28	0	28	0
C4	31	41	41	0	45	9.75	43	4.87	46	10.8	41	0	41	0	41	0
C5	37	44	44	0	48	9.09	48	9.09	49	10.2	44	0	44	0	44	0
C6	41	49	49	0	52	6.12	52	6.12	56	12.5	49	0	49	0	49	0
D1	34	39	39	0	46	19.44	45	15.38	47	17	39	0	40	2.56	39	0
D2	41	50	50	0	56	12	54	8	58	16	50	0	50	0	50	0
E1	46	55	55	0	59	7.27	59	7.27	62	12	55	0	55	0	55	0
E2	61	77	77	0	85	10.38	82	6.49	91	15.3	778	0	79	2.59	77	0
F	59	75	75	0	80	6.66	81	8	84	10.7	75	0	76	1.33	75	0
Average			-	1.25	-	10	-	9.97	-	14.27	-	0	-	0.51	-	0

Table 7: Numerical results of CP, heuristics, and meta-heuristics methods using ToSP data-sets for MCT-PSDPP (300-second runs)

Id	Id/C^*_{max}	CP		ECM		CGM		LS-standard		LS-split		SA		BS	
		C	G(%)	C	G(%)	C	G(%)	C	G(%)	C	G(%)	C	G(%)	C	G(%)
1	257	257	0	257	0	257	0	257	0	257	0	257	0	257	0
2	257	257	0	257	0	257	0	257	0	257	0	257	0	257	0
3	177	177	0	179	1.13	187	5.65	191	7.3	177	0	188	6.21	177	0
4	225	225	0	226	0.44	238	5.78	281	19.9	225	0	235	4.44	225	0
5	361	361	0	361	0	364	0.83	388	6.9	361	0	363	0.55	361	0
6	147	147	0	154	4.76	153	4.08	154	4.5	147	0	153	4.08	147	0
7	114	114	0	126	10.53	136	19.30	114	0	115	0.87	125	9.64	114	0
8	245	245	0	252	2.86	263	7.35	256	3.9	245	0	250	1.62	246	0.8
9	1187	1187	0	1235	4.04	1323	11.46	1187	0	1192	0.42	1318	11.03	1187	0
10	2273	2273	0	2273	0	2423	6.6	2275	0	2273	0	2331	2.55	2273	0
11	878	878	0	949	8.09	917	4.44	878	0	890	1.36	1089	24.03	878	0
12	3354	3595	6	3638	8.47	3560	6.14	3354	0	3668	0.76	4394	31.01	3354	0
Average		-	0.5	-	3.36	-	5.96	-	3.33	-	0.28	-	7.93	-	0

Table 8: Numerical results of CP, heuristics, and meta-heuristics methods using MMOpt kernels for MCT-PSDPP (300-second runs)

upper bound, in terms of line plots. Since the Simulated Annealing `SA` uses random values, we run it 10 times and get the average time where each C_{max} is reached in figures 11 and 12. Additionally, we observed that the `SA` reaches the same quality solutions in approximately the same time (less than 10 seconds). It is dominated by both the beam search algorithm and the "split LocalSolver model". We, therefore, only run it once on all the datasets we consider. We show the convergence on two instances: instance E2 (from ToSP data-sets: datE2) and Kernel 12 (from MMOpt kernels: Cameleon Sd), since the results were similar for all instances.

11.4.3 Discussions

As illustrated in Tables 6 and 7, we can see that the Beam Search `BS` can solve optimally all the instances in the ToSP benchmark (instances goes from size 9×10 to 40×60) in a few milliseconds. The second LocalSolver model `LS-split`

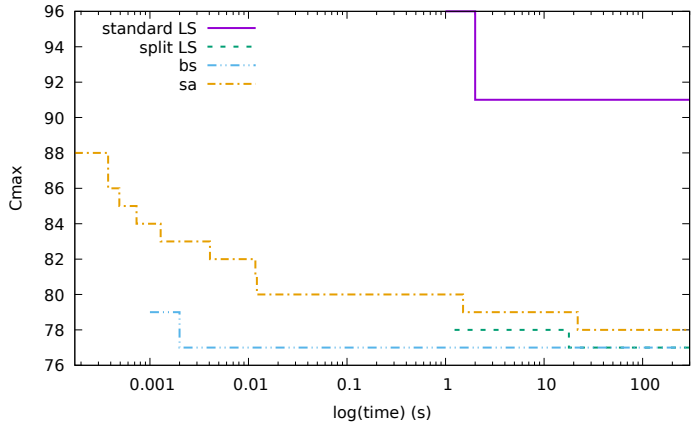


Fig. 11: Convergence of `LS-standard`, `LS-split`, `SA`, and `BS` methods for instance E2 (from ToSP data-sets: datE2)

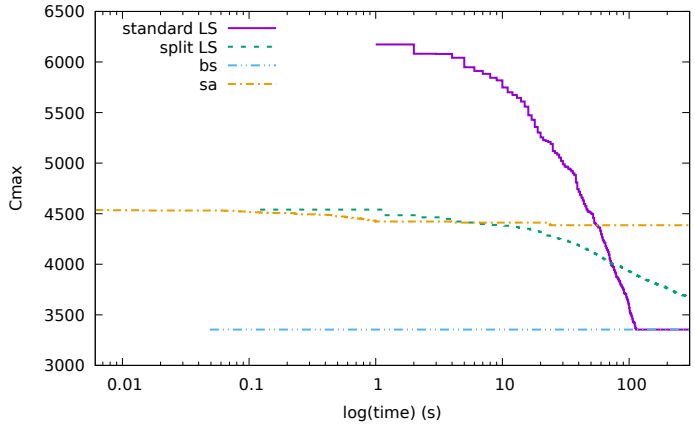


Fig. 12: Convergence of `LS-standard`, `LS-split`, `SA`, and `BS` methods for Kernel 12 (from MMOpt kernels: Cameleon Sd)

also obtains very good results. Similarly, the Constraint Programming `CP` can find the proof of optimality, except the instance E2 (we get the proof in **465** seconds). We noted that instances with a smaller density (like the instance E2) take more time than denser ones (like the instance F). The Simulated Annealing `SA` gives good results on most instances, with an average gap to the C^*_{max} of 0.51%. From Table 5, we can also see that the C_{max} provided by the two LocalSolver models (`LS-standard`, `LS-split`) is in average closer to the value of $lb_{C_{max}}$ than the different values given by each of the other algorithms. Both `ECM` and `CGM` algorithms provide relatively good upper bounds in a very short computation time.

In contrast, in the case of bigger instances of MMOpt (greater than 64×64 input/output tiles), we can see in both Table 8 and Table 6 that the Simulated Annealing `SA` presents huge gaps (around 40% on some instances). On the other hand, the Beam Search `BS` can find the optimal value for 11 instances in a few milliseconds, except the 8th one (Polar kernel), where it gets stuck very close to the optimal value (246 instead of 245). The Constraint Programming `CP` gives the optimal C_{max} for all instances in less than 22 seconds on average, except the 12th one (Cameleon Sd kernel where we need **8000** seconds to find the proof of optimality). We remark that the split model of LocalSolver overall performs better than the standard model. Indeed, it can find the optimal value on 8 over 12 instances, and for the 4 others, the gap is negligible (respectively 0.87%, 0.42%, 1.36%, and 0.76%). Finally, we obtain gaps of 3.36%, resp. 5.96% for both `ECM` and `CGM` algorithms, they will still be able to provide relatively good upper bounds even on big instances.

From both Fig. 10 and Fig. 11, we may find that (i) the Beam Search `BS` finds optimal solutions in a few milliseconds (or on the only instance it does not, it provides good solutions). (ii) the Simulated Annealing `SA` performs well on small instances, but it is far behind big instances. (iii) the LocalSolver standard model presents high gaps for most instances (around 20%). The split version around less than 1.5%.

In summary, these numerical experiments show that the Beam Search `BS` and the Constraint Programming `CP` perform better on both ToSP and MMOpt benchmarks than all other proposed algorithms for solving the MCT-PSDPP.

12 Conclusion and Future Work

In this paper, we studied a non-classical scheduling problem MCT-PSDPP (Minimum Completion Time of 3-PSDPP). This problem is defined as a variant of the Job Shop Scheduling Problem with tooling constraints, in which the completion time (makespan) is to be minimized. Diverse solution methods, including three mathematical programming models and four sets of algorithms (Constraint Programming, LocalSolver, Simulated Annealing, and Beam Search), have been applied to tackle this optimization problem. Computational results on two sets of benchmarks have been reported and analyzed. Globally, MIP models fail to find good bounds. In fact, the `MCT-3b` appears to be more convenient than the other models. On the other hand, Constraint Programming seems to be able to handle well this kind of problem. Besides, the Beam Search algorithm seems to be an interesting choice since it finds optimal solutions very quickly and can be implemented easily on embedded vision systems.

Further research may focus on scheduling problems with other objectives and/or other constraints related to some input parameters (i.e., prefetch time α, computation time β, number of prefetches N, number of buffers Z, ...).

References

1. Ibm ilog cplex optimization. http://ibmdecisionoptimization.github.io/docplex-doc/cp/index.html
2. Tool switching problem benchmarks. http://www.unet.edu.ve/~jedgar/ToSP/ToSP.htm
3. Tool switching problem visualization. http://librallu.gitlab.io/hypergraph-viz/
4. Al-Fawzan, M., K.S., A.S.: A tabu search based algorithm for minimizing the number of tool switches on a flexible machine. Computers & Industrial Engineering **44**(1), 35–47 (2003)
5. Amaya, J., Cotta, C., Fernández, A.: A memetic algorithm for the tool switching problem. In: Hybrid metaheuristics, pp. 190–202. Springer (2008)
6. Arbib, C., Flammini, M., Nardelli, E.: How to survive while visiting a graph. Discrete applied mathematics **99**(1-3), 279–293 (2000)
7. Bard, J.: A heuristic for minimizing the number of tool switches on a flexible machine. IIE Transactions **20**(4), 382–391 (1988)
8. Bellas, N., Chai, S., Dwyer, M., Linzmeier, D.: Real-time fisheye lens distortion correction using automatically generated streaming accelerators. In: 17th IEEE Symposium on Field Programmable Custom Computing Machines, FCCM'09., pp. 149–156 (2009)
9. Benoist, T., Estellon, B., Gardi, F., Megel, R., Nouioua, K.: Localsolver 1. x: a black-box local-search solver for 0-1 programming. 4or **9**(3), 299 (2011)
10. Calmels, D.: The job sequencing and tool switching problem: state-of-the-art literature review, classification, and trends. International Journal of Production Research **0**(0), 1–21 (2018)
11. Catanzaro, D., Gouveia, L., Labbé, M.: Improved integer linear programming formulations for the job sequencing and tool switching problem. European Journal of Operational Research **244**(3), 766–777 (2015)
12. Cherroun, H., Darte, A., Feautrier, P.: Reservation table scheduling: branch-and-bound based optimization *vs.* integer linear programming techniques. RAIRO - Operations Research **41**(4), 427–454 (2007)
13. Crama, Y., Kolen, A., Oerlemans, A., Spieksma, F.: Minimizing the number of tool switches on a flexible machine. International Journal of Flexible Manufacturing Systems **6**(1), 33–54 (1994)
14. Feautrier, P.: Parametric integer programming. Revue française d'automatique, d'informatique et de recherche opérationnelle **22**(3), 243–268 (1988)
15. Garey, M., Johnson, D., Sethi, R.: The complexity of flowshop and jobshop scheduling. Mathematics of operations research **1**(2), 117–129 (1976)
16. Gendreau, M., Potvin, J.: Handbook of metaheuristics, vol. 2. Springer (2010)
17. Graham, R., Lawler, E., Lenstra, J., Rinnooy Kan, A.: Optimization and approximation in deterministic sequencing and scheduling: a survey. Annals of discrete mathematics **5**, 287–326 (1979)
18. Hadj Salem, K., Kieffer, Y., Mancini, M.: Formulation and practical solution for the optimization of memory accesses in embedded vision systems. In: Proceedings of the 2016 Federated Conference on Computer Science and Information Systems, FedCSIS 2016, Gdańsk, Poland, September 11-14, 2016, pp. 609–617 (2016)
19. Hadj Salem, K., Kieffer, Y., Mancini, M.: Meeting the Challenges of Optimized Memory Management in Embedded Vision Systems Using Operations Research, pp. 177–205. Springer International Publishing (2018)
20. Hertz, A., Laporte, G., Mittaz, M., Stecke, K.: Heuristics for minimizing tool switches when scheduling part types on a flexible machine. IIE transactions **30**(8), 689–694 (1998)
21. Laporte, G., Salazar-Gonzalez, J., Semet, F.: Exact algorithms for the job sequencing and tool switching problem. IIE Transactions **36**(1), 37–45 (2004)
22. Lawler, E., Lenstra, J., Rinnooy Kan, A., Shmoys, D.: Sequencing and scheduling: Algorithms and complexity. Handbooks in operations research and management science **4**, 445–522 (1993)
23. Libralesso, L., Bouhassoun, A.M., Cambazard, H., Jost, V.: Tree search algorithms for the sequential ordering problem. arXiv preprint arXiv:1911.12427 (2019)

24. Libralesso, L., Jost, V., Hadj Salem, K., Fontan, F., Maffray, F.: Triangle width: at the intersection of graph theory, scheduling and matrix visualization (2019). Preprint submitted to Annals of Operations Research
25. Mancini, S., Rousseau, F.: Enhancing non-linear kernels by an optimized memory hierarchy in a high level synthesis flow. In: Proceedings of the Conference on Design, Automation and Test in Europe, pp. 1130–1133. EDA Consortium (2012)
26. Ow, P., Morton, T.: Filtered beam search in scheduling. The International Journal Of Production Research **26**(1), 35–62 (1988)
27. Pinedo, M.: Scheduling: Theory, Algorithms, and Systems. Springer Publishing Company, Incorporated (2016)
28. Privault, C., Finke, G.: Modelling a tool switching problem on a single nc-machine. Journal of Intelligent Manufacturing **6**(2), 87–94 (1995)
29. Sabuncuoglu, I., Bayiz, M.: Job shop scheduling with beam search. European Journal of Operational Research **118**(2), 390–412 (1999)
30. Tang, C., Denardo, E.: Models arising from a flexible manufacturing machine, part i: Minimization of the number of tool switches. Operations Research **36**(5), 767–777 (1988)
31. Thornton, A., Sangwine, S.: Log-polar sampling incorporating a novel spatially variant filter to improve object recognition. In: Sixth International Conference on Image Processing and Its Applications, vol. 2, pp. 776–779 (1997)
32. Vidal, T.: Split algorithm in o (n) for the capacitated vehicle routing problem. Computers & Operations Research **69**, 40–47 (2016)
33. Viola, P., Jones, M.: Robust real-time face detection. International journal of computer vision **57**(2), 137–154 (2004)
34. Zhou, B., Xi, L., Cao, Y.: A beam-search-based algorithm for the tool switching problem on a flexible machine. The International Journal of Advanced Manufacturing Technology **25**(9–10), 876–882 (2005)