



UNIVERSITÉ
LAVAL

A yellow industrial robot arm is shown in a factory setting, positioned over a dark metal structure. The background is slightly blurred, showing other industrial equipment and a blue wall.

Anas Neumann

Ordonnancement d'une cellule robotisée

BOMBARDIER

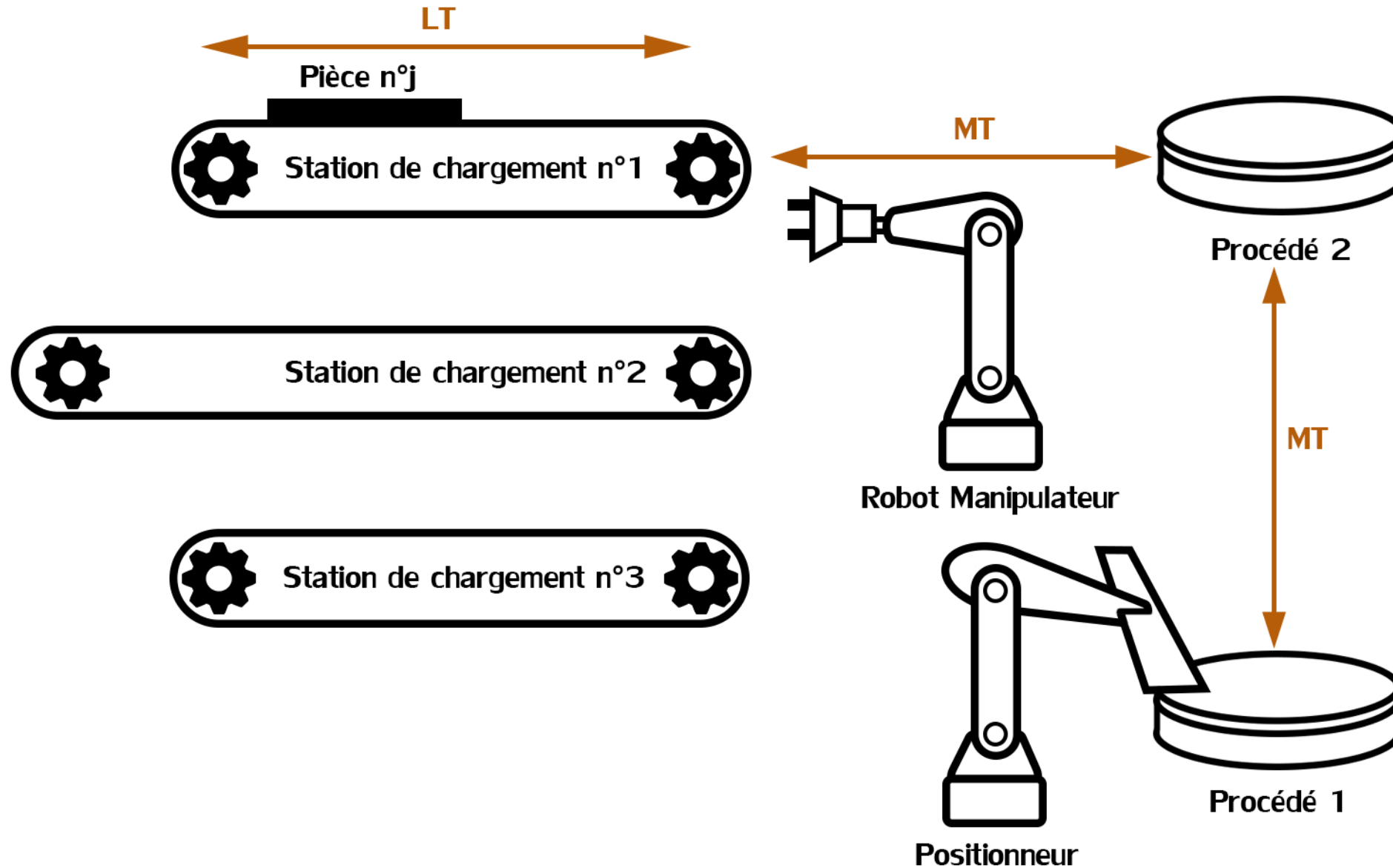
Plan

- I. Présentation du problème
- II. Proposition d'une méta-heuristique avec multiples améliorations
- III. Implémentation dans une application temps réel
- IV. Résolution par les mathématiques (modèle linéaire en nombres entiers)
- V. Analyse de la qualité et des performances calculatoires
- VI. Conclusion



Partie I Présentation du problème

Partie I : présentation du problème



Partie I : Les fichiers de données



.xls(x)

# pièce	tps procédé 1	tps procédé 2	longue pièce	date due	seq. 1
#1	6,00			6,5	1
#2		5,00	1	12,1	2
#3		2,00	1	3,5	2

seq. 2	seq. 3	positionnement	tps seq. 1	tps seq. 2	tps seq. 3	loaded	status
		0,5				1	3
		0,5				2	1
		0,5				0	0



Partie II

Proposition d'une méta-heuristique avec multiples améliorations

Partie II : Pseudocode

1^{ère} phase : construction d'une solution réalisable & performante

1. Trier les pièces selon leur date due (ordre croissant).
2. Pour deux dates équivalentes, trier les pièces dans l'ordre croissant de la durée des opérations à exécuter (afin de libérer la station au plus tôt).

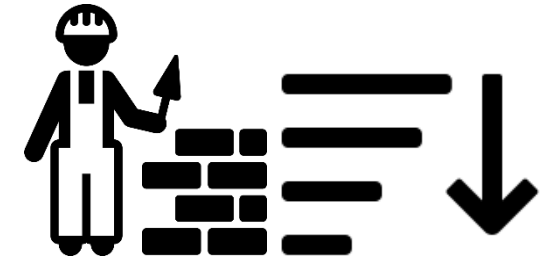
⇒ Il n'y aura ni parallélisme ni réalisation intercalée de deux pièces dans la solution de départ. Les pièces sont réalisées entièrement avant de passer à la suivante

2nd phase : permutation de l'ordre des opérations et parallélisme (amélioration n°1)

1. Pour toute opération non parallélisée, tester [*par "simulation"*] si un changement de mode réduit la valeur du retard. Si oui, effectuer ce changement de mode. Si le changement n'affecte pas le retard, favoriser le parallélisme afin de maximiser l'utilisation (et réduire le makespan).
2. S'arrêter lors que l'on a effectué tous les parallélismes possibles. Exécuter la 3^{ème} phase uniquement si le retard n'est pas nul.

3^{ème} phase : permutation de l'ordre de deux pièces consécutives (amélioration n°2)

1. Tester [*par "simulation"*] si permuter l'ordre de deux pièces consécutives permet de raccourcir le retard en augmentant les traitements parallèles (*appliquer la seconde phase sur la nouvelle solution*). Si oui, effectuer ce changement.
2. S'arrêter lors ce que l'on a testé toutes les paires consécutives de pièces ou que le retard est nul.



TRI + CONSTRUCTION



GAIN EN PARALLELISME



RECHERCHE LOCALE

Partie II : Simulation

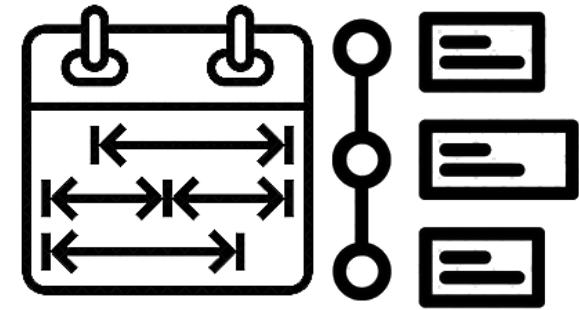
$LT \leftarrow$ durée de chargement/déchargement d'une pièce sur une station de chargement

$MT \leftarrow$ durée d'un mouvement unique du robot

1. $date \leftarrow$ durée des mouvements du robot pour sortir pièces non urgentes (MT pour une unique pièce, $3 \times MT$ pour 2 pièces)
2. Initialiser des stations à l'état "libre" selon les déchargements effectués (0 , LT , $MT+LT$ ou $3 \times MT + LT$)
3. Répéter pour chaque pièce (selon l'ordre choisi)
 - a. Charger la pièce dans la 1^{ère} station disponible possible (selon la taille de la pièce) au temps "date" + LT . Si la pièce est déjà chargée, ne rien faire.
 - b. Répéter pour chaque opération
 - Exécuter au plus tôt selon la valeur de "date", la disponibilité du robot (1 à 3 mouvements nécessaires selon le mode de la dernière opération de la précédente pièce), et la durée de mise sur le positionneur (en cas de mode B sans historique).
 - Si l'opération (de procédé 2) se fait en parallèle d'une opération en mode B, recalculer les dates de la pièce bloquée en mode B car elle pourrait devoir attendre la fin de l'opération de procédé 2 (*relancer la simulation*).
 - c. Sortir la pièce (compter les mouvements robot 1 à 4) et libérer la station au nouveau temps "date" + LT .
 - d. Calculer le retard pour cette pièce par soustraction avec sa "date due"

Fin Répéter

Fin Répéter



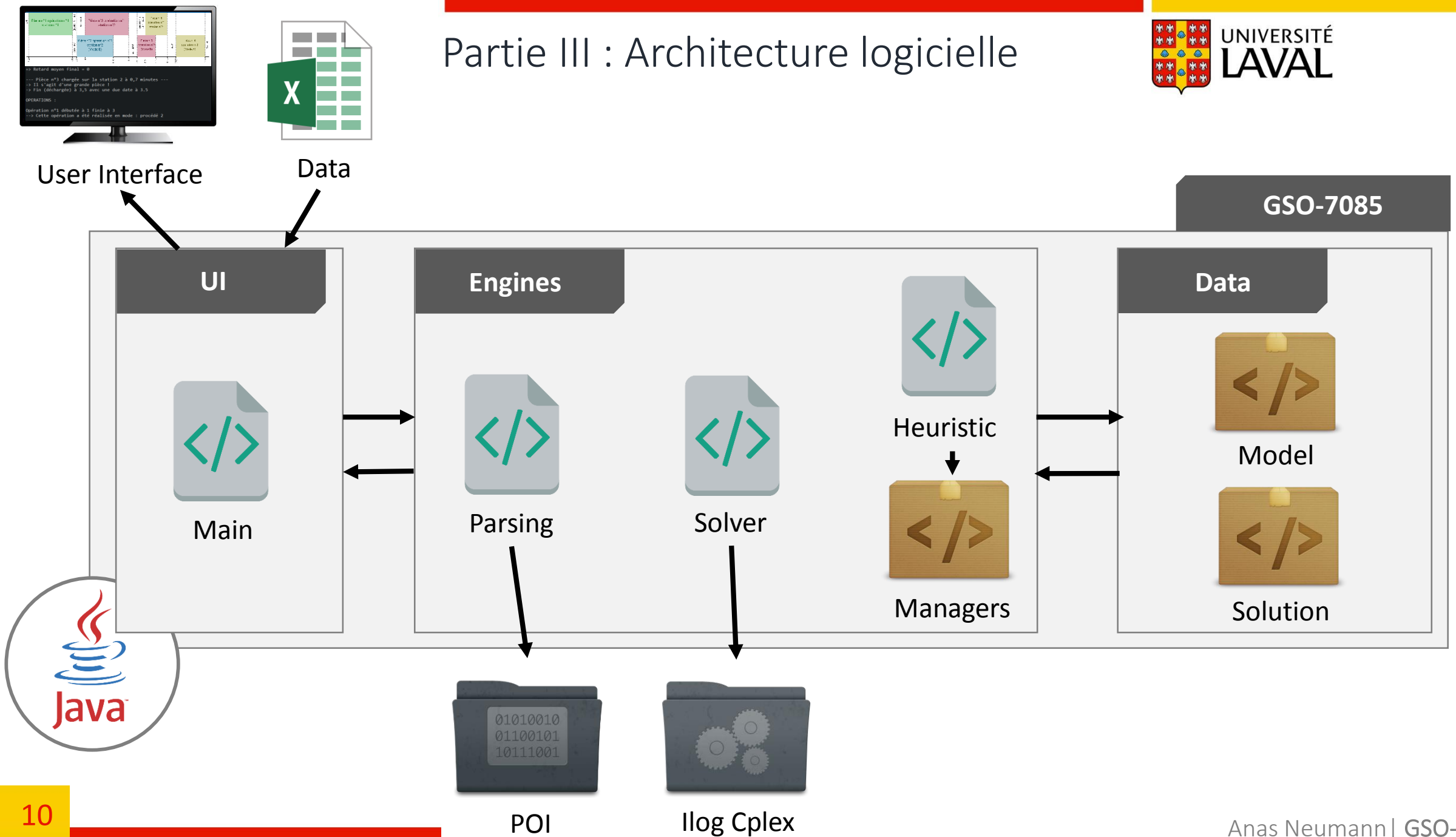
SIMULATION DES PIECES



Partie III

Implémentation dans une application temps réel

Partie III : Architecture logicielle



Partie III : L'heuristique

```
/**
 * Main method
 * @param args
 */
public static void main(String[] args) {
    Double MT = 0.3, LT = 0.2;

    Instance problem1 = ParsingFileEngine.BuildInstance("instance_3.xlsx");
    OptimalSchedulingEngineV2.getEngine().solve(problem1);
    MultiNeighborhoodSearchEngine.getEngine().solve(problem1, MT, LT);

    Instance problem2 = ParsingFileEngine.BuildInstance("instance_4_2.xlsx");
    OptimalSchedulingEngineV2.getEngine().solve(problem2);
    MultiNeighborhoodSearchEngine.getEngine().solve(problem2, MT, LT);

    Instance problem3 = ParsingFileEngine.BuildInstance("instance_dynamique.xlsx");
    OptimalSchedulingEngineV3.getEngine().solve(problem3);
    MultiNeighborhoodSearchEngine.getEngine().solve(problem3, MT, LT);

    Instance problemReel = ParsingFileEngine.BuildInstance("instanceBig2.xlsx");
    MultiNeighborhoodSearchEngine.getEngine().solve(problemReel, MT, LT);
}
```

```
/**
 * Solve the problem
 * @param problem
 * @param MT
 * @param LT
 */
public Solution solve(Instance problem, Double MT, Double LT) {
    Long startTime = System.currentTimeMillis();
    Solution s = buildInitialSolution(problem, MT, LT);
    for(int index = 0; index < s.getJobs().size() - 1; index++) {
        if(s.computeAverageDelay().equals(0.0)) {
            displaySolution(s, startTime, System.currentTimeMillis());
            return s;
        }
        Solution newSolution = localSearch(s, index);
        if(newSolution.computeAverageDelay() <= s.computeAverageDelay()) {
            s = newSolution;
        }
    }
    displaySolution(s, startTime, System.currentTimeMillis());
    return s;
}
```


Partie III : Les phases d'amélioration



```
/**
 * Améliorer une solution en cherchant à y intégrer du parallélisme
 * @param s
 * @return
 */
public Solution amelioration(Solution s) {
    Double currentDelay = s.computeAverageDelay();
    Double newDelay = 0.0;
    for(CeduledJob j : s.getJobs()) {
        for(CeduledOperation o : j.getCeduledOperations()) {
            if(o.getMode().equals(CeduledOperation.MODE_A)) {
                o.setMode(CeduledOperation.MODE_B);
                newDelay = runSolution(s).computeAverageDelay();
                if(currentDelay < newDelay) {
                    o.setMode(CeduledOperation.MODE_A);
                } else {
                    currentDelay = newDelay;
                }
            }
        }
    }
    return runSolution(s);
}
```

```
/**
 * chercher à intervenir deux pièces à partir d'une position
 * @param s
 * @param index
 * @return
 */
public Solution localSearch(Solution s, int index) {
    Solution neighborhood = s.clone();
    CeduledJob j1 = neighborhood.getJobs().get(index);
    CeduledJob j2 = neighborhood.getJobs().get(index + 1);
    neighborhood.getJobs().add(index, j2);
    neighborhood.getJobs().add(index + 1, j1);
    neighborhood.getJobs().remove(index + 2);
    neighborhood.getJobs().remove(index + 2);
    return amelioration(runSolution(neighborhood));
}
```

```
/**
 * Dérouler une solution et calculer tous les temps
 * @param s
 * @return the solution with informations
 */
public Solution runSolution(Solution s) {
    StationManager m = StationManager.createManager();
    DateManager d = DateManager.createManager(s.MT);
    ParallelManager p = ParallelManager.createManager();
    removeJobs(s,d,m);
    for(CeduledJob j : s.getJobs()) {
        d.date = Math.max(m.enterStation(j, s.LT), d.date);
        runJobFromOperation(m, j, 0, d, s, p);
    }
    return s;
}
```

```
/**
 * Un job souhaite entrer dans une station
 * @return
 */
public Double enterStation(CeduledJob job, Double LT) {
    if(!job.isRemoved() && job.getLoadingHistory() > 0) {
        job.setLoadedDate(0.0);
        job.setLoadedStation(job.getLoadingHistory()-1);
    } else {
        int station = 1;
        Double dateLoad = loadingStations.get(1);
        if(!job.isSize()) {
            for(Double s : loadingStations) {
                if(s < dateLoad) {
                    station = loadingStations.indexOf(s);
                    dateLoad = s;
                }
            }
        }
        job.setLoadedDate(dateLoad + LT);
        job.setLoadedStation(station);
    }
    return job.getLoadedDate();
}
```

Partie III : La simulation

```
public void runJobFromOperation(StationManager m, CeduledJob j, int index, DateManager d, Solution s, ParallelManager p) {
    for(int i=index; i<j.getCeduledOperations().size(); i++) {
        CeduledOperation o = j.getCeduledOperations().get(i);
        if(!alreadyReady(j, index)) {
            d.doAMove();
        }
        if(o.getMode().equals(CeduledOperation.MODE_B)) {
            if(!alreadyReady(j, index)) {
                d.date += j.getPositionTime();
            }
            p.bloc(j, o);
        }
        if(o.getMode().equals(CeduledOperation.MODE_C) && (i == 0) && null != p.job && !j.getId().equals(p.job.getId())) {
            d.date = Math.max(j.getLoadedDate(), p.operation.getBeginDate());
            if(alreadyReady(p.job, p.job.getCeduledOperations().indexOf(p.operation))) {
                d.doAMove();
            } else {
                d.moves(2);
            }
            o.setBeginDate(d.date);
            o.setEndDate(d.date + o.getProcessingTime());
            d.date = Math.max(p.operation.getEndDate(), o.getEndDate());
            if(i >= (j.getCeduledOperations().size() - 1)) {
                m.leaveStation(j, o.getEndDate() + s.MT, j.getLoadedStation(), s.LT);
                d.date = Math.max(o.getEndDate() + (2 * s.MT), d.date);
                runJobFromOperation(m, p.job, p.job.getCeduledOperations().indexOf(p.operation)+1, d, s, p);
                p.job = null;
                p.operation = null;
                return;
            } else {
                d.moves(2);
                runJobFromOperation(m, p.job, p.job.getCeduledOperations().indexOf(p.operation)+1, d, s, p);
                p.job = null;
                p.operation = null;
            }
        }
    }
}
```




Partie IV

Modèle mathématique dynamique en nombres entiers

Notation

- n : nombre de pièces dans une instance à résoudre
- $w = \{1, 2\}$: un procédé de soudure
- $j = 1, \dots, n$: une pièce
- $q \in \mathbb{R}^+$: une position dans une séquence d'opérations (une ou plusieurs par pièce)
- $l = \{1, 2, 3\}$: une station de chargement (considérées comme des machines parallèles)
- $m = \{1, 2, 3\}$: un mode de soudure (A, B ou C respectivement)

Données constantes

- $t_{jq} \in \mathbb{R}^+$, la durée de soudure d'une opération n° q d'une pièce j
- $a_{jqw} = \{0, 1\}$, vaut :
 - 1 si une pièce j doit être soudée par le procédé w durant l'opération n° q
 - 0 sinon
- $s_j = \{0, 1\}$, la taille d'une pièce 1 = longue, 0 = petite ou moyenne
- $d_j \in \mathbb{R}^+$, la date due d'une pièce j
- $p_j \in \mathbb{R}^+$, le temps que prend une pièce j à être placée sur le positionneur (mode B)
- $I = \sum_{j=1}^n \sum_q (t_{jq} + p_j + 3MT + 2LT)$, (*infinity*) une borne supérieure

- $lh_{jl} = \{0, 1\}$, (*loading history*) vaut :
 - 1 si une pièce j est au départ dans la station de chargement l
 - 0 sinon
- $ph_j = \{0, 1\}$, (*position history*) vaut :
 - 1 si une pièce j est au départ sur le positionneur
 - 0 sinon
- $wh_j = \{0, 1\}$, (*welding history*) vaut :
 - 1 si une pièce j est au départ tenue par le robot
 - 0 sinon
- $LT \in \mathbb{R}^+$, (*loading time*) la durée de chargement ou déchargement d'une pièce sur l'une des stations.
- $MT \in \mathbb{R}^+$, (*move time*) la durée d'un mouvement du robot.

Variables de décision

- $L_{jl} = \{0, 1\}$: (*Load*) vaut :
 - 1 si une pièce j sera chargée dans une station l
 - 0 sinon
- $B_{jl} \in \mathbb{R}^+$: (*Begin*) date d'entrée d'une pièce j dans une station l
- $M_{jqm} = \{0, 1\}$, vaut :
 - 1 si une opération n° q d'une pièce j sera exécutée en mode m
 - 0 sinon
- $W_{jq} \in \mathbb{R}^+$: (*Weld*) date de début d'une opération n° q d'une pièce j
- $P_{ikjq} = \{0, 1\}$, (*Precedes*) vaut :
 - 1 si une opération n° k d'une pièce i sera exécutée (soudée) avant une opération n° q d'une pièce j
 - 0 sinon

- $D_j \in \mathbb{R}^+$: (*Delay*) retard réel d'une pièce j
- $PA_{jq} = \{0,1\}$: (*Parallel Action*) vaut :
 - 1 si une opération q de type 3 d'une pièce j est exécutée en parallèle avec une en mode B.
 - 0 sinon
- $R_{jl} = \{0,1\}$: (*Remove*) vaut :
 - 1 si une pièce j sera déchargée de la station l
 - 0 sinon

Fonction objectif :
$$\text{Min } \frac{1}{n} \times \sum_{j=1}^n D_j$$

Sous contraintes

$$[C_1] : P_{ikjq} + P_{jqik} = 1 \quad \forall i=1,\dots,n; \quad \forall j=1,\dots,n; \quad \forall k \in \mathbb{R}^+; \quad \forall q \in \mathbb{R}^+; \quad i \neq j$$

$$[C_2] : W_{jq} - \text{end}(j, q-1) - MT - 3MT * PA_{j(q-1)} \geq 0; \quad \forall q \geq 2; \quad \forall j = 1, \dots, n$$

$$[C_3] : W_{jq} - \text{end}(i, k) - 2MT + I(1 - P_{ikjq}) + IM_{ik2} \geq 0$$
$$\forall i=1,\dots,n; \quad \forall j=1,\dots,n; \quad \forall k \in \mathbb{R}^+; \quad \forall q \in \mathbb{R}^+; \quad i \neq j$$

$$[C_4] : W_{jq} - \text{end}(i, k) - 2MT + I(1 - P_{ikjq}) + IM_{jq3} \geq 0$$
$$\forall i=1,\dots,n; \quad \forall j=1,\dots,n; \quad \forall k \in \mathbb{R}^+; \quad \forall q \in \mathbb{R}^+; \quad i \neq j$$

$$[C_5] : W_{jq} - W_{ik} - (M_{ik2} \times p_i) - 2MT + I(1 - P_{ikjq}) \geq 0$$
$$\forall i=1,\dots,n; \quad \forall j=1,\dots,n; \quad \forall k > 1; \quad \forall q \in \mathbb{R}^+; \quad i \neq j$$

$$[C_5] : W_{jq} - W_{ik} - (M_{ik2} \times p_i) - 2MT + I(1 - P_{ikjq}) \geq 0$$

$$\forall i=1, \dots, n; \forall j=1, \dots, n; \forall k > 1; \forall q \in \mathbb{R}^+; i \neq j$$

$$W_{jq} - W_{ik} - ((M_{ik2} \times p_i) - 2MT) \times (1 - ph_i) + I(1 - P_{ikjq}) \geq 0$$

$$\forall i=1, \dots, n; \forall j=1, \dots, n; \forall k=1; \forall q \in \mathbb{R}^+; i \neq j$$

$$[C_6] : W_{j1} - MT - \sum_{l=1}^3 (B_{jl} - MT \times (1 - R_{jl}) \times (ph_j + wh_j) \times lh_{jl}) \geq 0 \quad \forall j=1, \dots, n;$$

$$[C_7] : \sum_{m=1}^3 M_{jqm} = 1; \quad \forall q \in \mathbb{R}^+; \quad \forall j = 1, \dots, n$$

$$[C_8] : M_{jq3} = a_{jq2}; \quad \forall q \in \mathbb{R}^+; \quad \forall j = 1, \dots, n$$

$$[C_9] : \sum_{l=1}^3 L_{jl} = 1; \quad \forall j = 1, \dots, n$$

$$[C_{10}] : L_{j2} - s_j \geq 0 ; \forall j = 1, \dots, n$$

$$[C_{11}] : B_{jl} - \text{end}(i, q) - 2LT - MT + \text{prec}(i, j, l) \geq 0$$

$$\forall j = 1, \dots, n; \forall i = 1, \dots, n; \forall l \in \{1, 2, 3\}; \forall q \in \mathbb{R}^+; i \neq j$$

$$[C_{12}] : B_{jl} - \text{free}(i, q, y, k) - 2LT - 3MT + \text{prec}(i, j, l) \geq 0$$

$$\forall y = 1, \dots, n; \forall j = 1, \dots, n; \forall i = 1, \dots, n; \forall l \in \{1, 2, 3\}; \forall k \in \mathbb{R}^+;$$
$$y \neq j \quad i \neq y \quad i \neq j$$

$$[C_{13}] : D_j \geq 0 \quad \forall j = 1, \dots, n$$

$$[C_{14}] : D_j - \text{end}(j, q) - LT - MT + d_j \geq 0 \quad \forall j = 1, \dots, n;$$

$$[C_{15}] : D_j - \text{free}(j, q, i, k) - LT - 3MT + d_j \geq 0 \quad \forall i = 1, \dots, n; \forall j = 1, \dots, n; \forall k \in \mathbb{R}^+; i \neq j$$

$$[C_{16}] : W_{jq} - end(i,k) - 2MT + I(1 - P_{ikjq}) + IPA_{jq} \geq 0$$

$$\forall i=1,\dots,n; \forall j=1,\dots,n; \forall k \in R^+; \forall q \in R^+; i \neq j$$

$$[C_{17}] : a_{jq2} - PA_{jq} \geq 0; \forall q \in R^+; \forall j = 1,\dots,n$$

$$[C_{18}] : B_{jl'} - \sum_{l=1}^3 R_{jl} (2LT + MT \times wh_j + 3MT \times ph_j) + I(1 - L_{jl'}) \geq 0$$

$$\forall j = 1,\dots,n; \forall l' = 1,2,3;$$

$$[C_{19}] : B_{jl} - lh_{il} (2LT + MT \times wh_i + 3MT \times ph_i) + I(1 - L_{jl}) \geq 0$$

$$\forall j = 1,\dots,n; \forall i = 1,\dots,n; \forall l \in \{1,2,3\}; i \neq j$$

$$[C_{20}] : R_{jl} + I(1 - lh_{jl}) + I(1 - P_{iljl}) + I(1 - L_{il}) \geq 1$$

$$\forall j = 1,\dots,n; \forall i = 1,\dots,n; \forall l \in \{1,2,3\}; i \neq j$$

$$[C_{21}] : W_{jl} - \sum_{i=1}^n \sum_{l=1}^3 R_{il} (MT \times wh_i + 2MT \times ph_i) \geq 0 \quad \forall j = 1,\dots,n;$$

- $prec(i,j,l) = I(1 - P_{iljl}) + I(1 - L_{il}) + I(1 - L_{jl})$
- $end(j,q) =$
 - **Si** $q > 1 : W_{jq} + t_{jq} + (M_{jq2} \times p_j)$
 - **Sinon** : $W_{jq} + t_{jq} + (M_{jq2} \times p_j) \times (1 - ph_j)$
- $free(j,q,i,k) =$
 - **Si** $n=2 : end(i,k) - I(1 - P_{jqik}) - I(1 - M_{jq2}) - I(1 - M_{ik3})$
 - **Sinon** : $end(i,k) - I(1 - P_{jqik}) - I(1 - M_{jq2}) - I(1 - M_{ik3}) - I(1 - PA_{ik}) - \sum_{y=1}^n \sum_v$
 $I \times a_{yv1} \times (P_{jqyv} + P_{yvik} - P_{jqik}) \quad y \neq j; \quad y \neq i$

Partie IV : Résolution avec Cplex

```
/**
 * Solve the problem with Cplex
 *
 * @param problem
 */
public void solve(Instance problem) {
    try {
        IloCplex cplex = new IloCplex();
        Model m = new Model(problem, true);
        createVariables(cplex, m);
        createObjectiveFunction(cplex, m);
        createConstraints(cplex, m);
        cplex.exportModel("lpex1.lp");
        if (cplex.solve()) {
            displayResult(cplex, m);
        }
        cplex.end();
    } catch (IloException e) {
        e.printStackTrace();
    }
}
```

```
private void createVariables(IloCplex cplex, Model m) throws IloException {
    String[] namesDelay = new String[m.nbrJobs];
    for (int j = 0; j < m.nbrJobs; j++) {
        namesDelay[j] = "D" + j;
        Integer nbrOpJ = m.jobs.get(j).getOperations().size();
        String[] namesLoads = new String[m.nbrLoadStations];
        String[] namesBegin = new String[m.nbrLoadStations];
        String[] namesRemove = new String[m.nbrLoadStations];
        String[] namesWeld = new String[nbrOpJ];
        String[] namesParallel = new String[nbrOpJ];
        m.varMode[j] = new IloIntVar[nbrOpJ][m.nbrModes];
        m.varPrecedence[j] = new IloIntVar[nbrOpJ][m.nbrJobs][];
        for (int q = 0; q < nbrOpJ; q++) {
            namesWeld[q] = "W(" + j + "," + q + ")";
            namesParallel[q] = "PA(" + j + "," + q + ")";
            String[] namesModes = new String[m.nbrModes];
            for (int o = 0; o < m.nbrModes; o++) {
                namesModes[o] = "M(" + j + "," + q + "," + o + ")";
            }
            for (int i = 0; i < m.nbrJobs; i++) {
                Integer nbrOpI = m.jobs.get(i).getOperations().size();
                String[] namesPrecedences = new String[nbrOpI];
                for (int k = 0; k < nbrOpI; k++) {
                    namesPrecedences[k] = "P(" + j + "," + q + "," + i + "," + k + ")";
                }
                m.varPrecedence[j][q][i] = cplex.boolVarArray(nbrOpI, namesPrecedences);
            }
            m.varMode[j][q] = cplex.boolVarArray(m.nbrModes, namesModes);
        }
    }
}
```

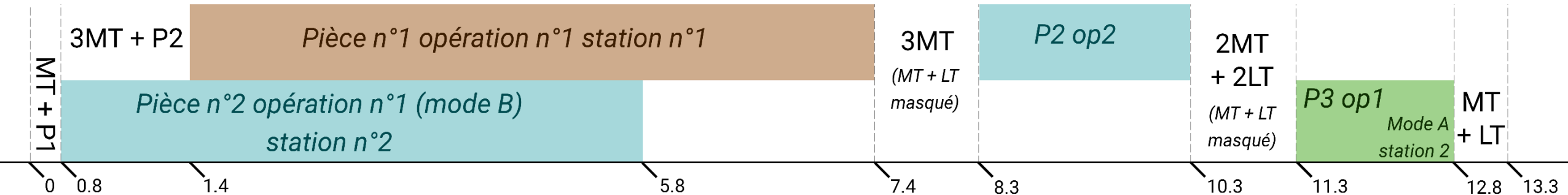



Partie V

Analyse de la qualité et de la performance calculatoire

Partie V : instance n°1

Pièce	Durée procédé 1	Durée procédé 2	Pièce longue ?	Date due	Opération n°1	Opération n°2	Temps de positionne ment
#1		6.00		7.90	2		0.5
#2	5.00	2.00	1(oui)	10.80	1	2	0.5
#3	1.50		1(oui)	13.30	1		0.5

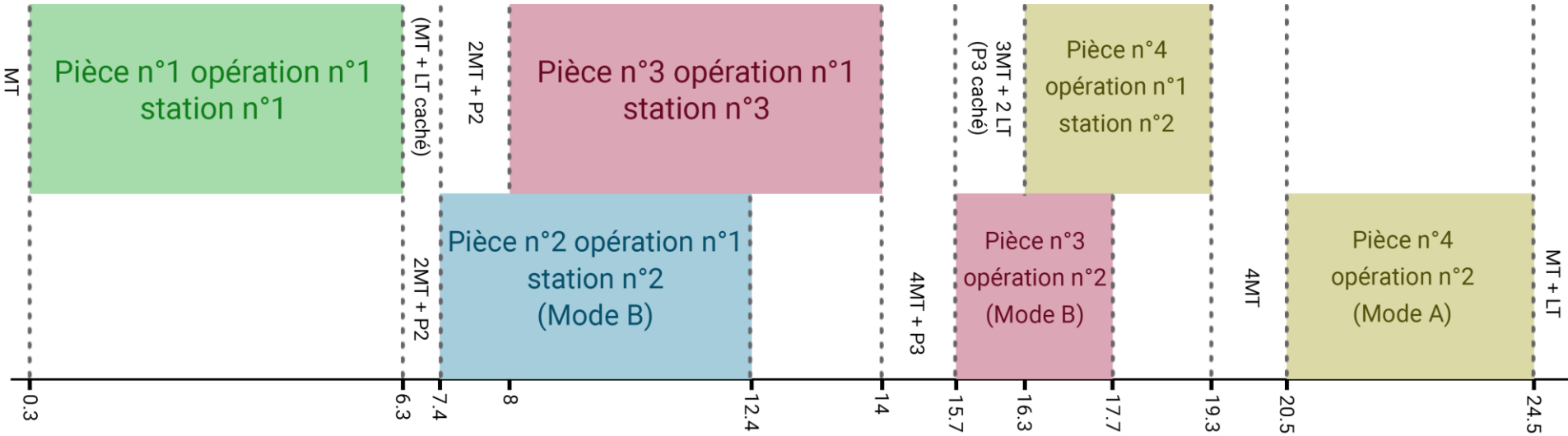


Partie V : instance n°2



1^{ère} AMELIORATION

Pièce	Durée procédé 1	Durée procédé 2	Pièce longue ?	Date due	Opération n°1	Opération n°2	Temps de positionne ment
#1		6.00		6.80	2		0.5
#2	5.00		1(oui)	15.10	1		0.5
#3	2.00	6.00		20.40	2	1	0.5
#4	4.00	3.00	1(oui)	25	2	1	0.5

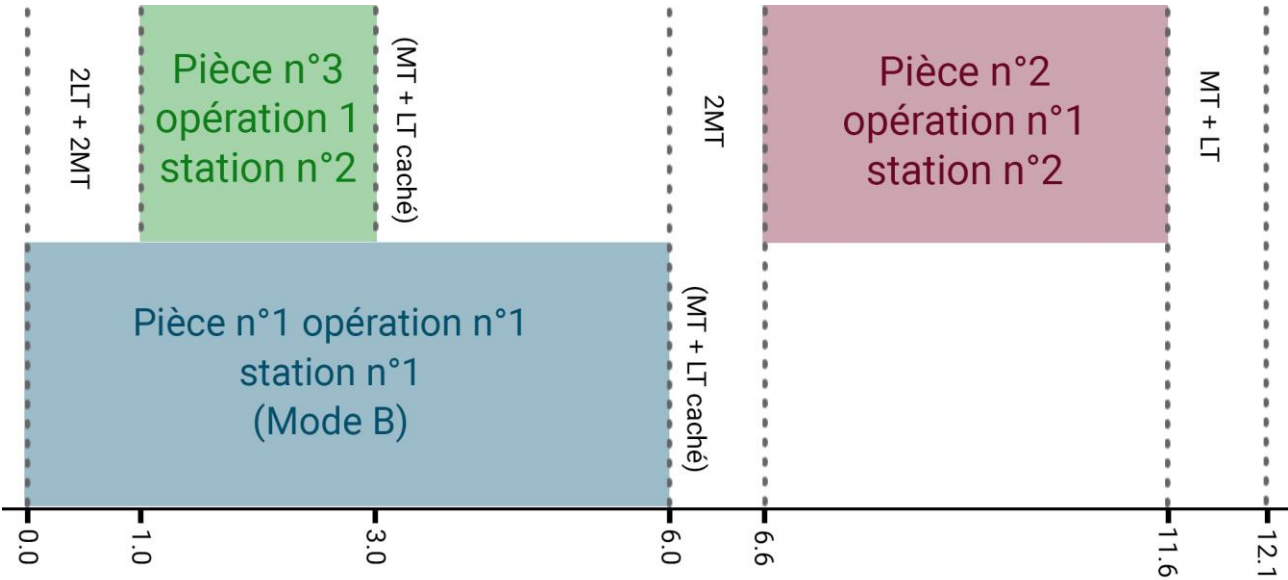


Partie V : instance n°3

Id	Durée opération restante	type d'opération	date due	Pièce longue ?	tps positionnement	station bloquée	status
# 1	6.00	1	06.50	non	0.50	1	3
# 2	5.00	2	12.10	oui	0.50	2	1
# 3	2.00	2	03.50	oui	0.50	-	0



2nd AMELIORATION + DECHARGMENT



Partie V : instance réelle

Nombre de pièces : **162**

Nombre d'opérations : **196**

→ **1 010 000 000** instructions environ pour l'heuristique
($196 \times 196 \times 162 \times 162 + 162 \times 162 \times 196 + 162 \times 196 + 162 + 196$)
= résolu en **0.99 secondes !**

→ **196!** instructions environ pour le modèle mathématique
(incalculable et incomparablement plus grand)

[13! Dépassant déjà le 6 x nombre d'opérations de l'heuristique]



TEMPS REEL



Partie VI Conclusion

Objectifs

- Déviation nulle dans la grande majorité des cas.
- Temps réel même avec un volume de données relativement grand.
- Complexité polynomiale $O(m^4)$.
- Application maintenable (organisée, taille réduite) et heuristique évolutive.

Améliorations

- Déviation en cas de possibilité d'enchaînement de parallélisme.
- Voisinage parfois pas assez grand (phase n°2).
- Heuristique très spécifique dans l'état actuel.

Merci de votre attention !