Model

Provided variables

- $nbrTasks$
- $nbrTasks$
- $nbrMachines$
- $nbrTools$
- $nbrTrays$
- $nbrFixtures$
- $nbrComponents$
- $nbrOutputs$
- $nbrConcurrentGroups$
- $nbrOrderedGroups$
- $tray_t$
- $output_t$
- $fixture_t$
- $componentsUsed_t$
- $mounting$
- $taking$
- $moving$
- $putting$
- $concurrentTasks_k$
- $order_k$
- $toolNeeded_t$
- $changeToolDuration_{tool_1,tool_2}$
- $usingMachine_t$
- d_t
- $taskSubComponents_t$
- $taskCompleteSubComponents_t$

Variables

$$nbrTasks$$

Figure 1: The number of tasks to schedule

$$nbrMachines$$

Figure 2: The number of machines available

$$nbrTools$$

Figure 3: The number of tools available

$$nbrTrays$$

Figure 4: The number of trays available

$$nbrFixtures$$

Figure 5: The number of fixtures available

$$nbrComponents$$

Figure 6: The number of components available

$$nbrOutputs$$

Figure 7: The number of outputs available

$$nbrConcurrentGroups$$

Figure 8: The number of concurrent groups used

$$nbrOrderedGroups$$

Figure 9: The number of ordered groups used

$$tasks = \{1 \dots nbrTasks\}$$

Figure 10: The tasks

$$machines = \{1 \dots nbrMachines\}$$

Figure 11: The machines

$$tools = \{1 \dots nbrTools\}$$

Figure 12: The tools

$$components = \{1 \dots nbrComponents\}$$

Figure 13: The components

$$trays = \{1 \dots nbrTrays\}$$

Figure 14: The trays

$$fixtures = \{1 \dots nbrFixtures\}$$

Figure 15: The fixtures

$$outputs = \{1 \dots nbrOutputs\}$$

Figure 16: The outputs

$$concurrentGroups = \{1 \dots nbrConcurrentGroups\}$$

Figure 17: The concurrent groups

$$orderedGroup_k \subset tasks,\ k \in \{1 \dots nbrOrderedGroups\}$$

Figure 18: Ordered group k

$$startTasks = \{nbrTasks + 1 \dots nbrTasks + nbrMachines\}$$

Figure 19: The start tasks. Is the predecessors to the first tasks for each arm

$$goalTasks = \{nbrTasks + nbrMachines + 1 \dots nbrTasks + nbrMachines \times 2\}$$

Figure 20: The goal tasks. Its the predecessors are the last tasks for each arm

$$allTasks = tasks \cup startTasks \cup goalTasks$$

Figure 21: All the tasks, including the start and goal tasks

$$tray_t \in trays \cup \{0\},\ t \in tasks$$

Figure 22: The tray task t uses. If none is used $tray_t = 0$

$$output_t \in outputs \cup \{0\},\ t \in tasks$$

Figure 23: The output task t uses. If none is used $output_t = 0$

$$fixture_t \in fixtures \cup \{0\},\ t \in tasks$$

Figure 24: The fixture task t uses. If none is used $fixture_t = 0$

$$componentsUsed_t \subset components,\ t \in tasks$$

Figure 25: The components used at task t

$$mounting \subset tasks$$

Figure 26: Set of tasks that mounts a component

$$taking \subset tasks$$

Figure 27: Set of tasks that takes a component

$$moving \subset tasks$$

Figure 28: Set of tasks that moves a component somewhere

$$putting \subset tasks$$

Figure 29: Set of tasks that puts a component somewhere

$$concurrentTasks_k \subset tasks,\ k \in concurrentGroups$$

Figure 30: Set of tasks needing concurrent execution

$$order_k \subset tasks,\ k \in orderedGroups$$

Figure 31: Array of tasks needed to be performed in a certain order on a single machine

$$order_{k,i} \in tasks,\ i \in \{1 \dots |orderedGroup_k|\},\ k \in orderedGroups$$

Figure 32: The i:th task to be performed i order according to the k:th $orderedGroup$

$$toolNeeded_t \in tools,\ t \in tasks$$

Figure 33: The tool needed for task t

$$changeToolDuration_{tool_1,tool_2} \in \{0 \dots 2^{32} - 1\},\ tool_1, tool_2 \in tools$$

Figure 34: The duration of changing from $tool_1$ to $tool_2$

$$putting_c \subset putting,\ c \in components$$

Figure 35: Set of tasks that puts component c somewhere

$$mounting_c \subset mounting,\ c \in components$$

Figure 36: Set of tasks that mounts component c

$$taking_c \subset taking,\ c \in components$$

Figure 37: Set of tasks that takes component c

$$moving_c \subset moving,\ c \in components$$

Figure 38: Set of tasks that moves component c somewhere

$$orderedSet = \bigcup_{\forall k \in orderedGroups} order_k,\ orderedSet \subset tasks$$

Figure 39: The set of tasks member of at least one ordered groups

$$usingMachine_t \in machines,\ t \in tasks$$

Figure 40: The machine task t uses

$$pred_t \in allTasks,\ t \in allTasks$$

Figure 41: The predecessor task of task t

$$d_t \in \{0 \dots 2^{32} - 1\},\ t \in tasks$$

Figure 42: The duration of task t

$$timeMatrixDepth = \frac{n^2 - n + 2}{2},\ n = nbrMachines$$

Figure 43: The depth of the 3D time matrix, i.e. the number of different transitions betweens different tools, independent of direction and a transition from tool to itself is the same transition for all tools

$$\begin{aligned} timeMatrix3D_{t_{from},t_{to},k} \in & \{0 \dots 2^{32} - 1\},\ t_{from} \in tasks \cup startTasks, \\ & t_{to} \in tasks,\ k \in \{0 \dots timeMatrixDepth\} \end{aligned}$$

Figure 44: The time to move from task t_{from} to task t_{to} changing tool according to k

$$\begin{aligned} maxE = & (max(\{d_t : t \in tasks\}) + \\ & max(\{timeMatrix3D_{t_1,t_2,k} : \forall t_1 \in tasks \cup startTasks, \\ & \forall t_2 \in tasks,\ \forall k \in \{0 \dots timeMatrixDepth\}\}) \times nbrTasks \end{aligned}$$

Figure 45: Rough upper limit of the total schedule time. Assumes all move times take as long as the longest move time existing in the schedule. And likewise for the task durations.

$$start_t \in \{0 \dots maxE\},\ t \in allTasks$$

Figure 46: The start time for task t

$$end_t = start_t + d_t,\ t \in allTasks$$

Figure 47: The end time for task t

$$makespan \in \{0 \dots maxE\}$$

Figure 48: The makespan for the whole schedule, the time to minimize

$$moveDuration_t \in \{0 \dots maxE\},\ t \in allTasks$$

Figure 49: The duration of the move to task t from its predecessor

$$moveStart_t \in \{0 \dots maxE\},\ t \in allTasks$$

Figure 50: The start time for the move to task t from its predecessor

$$moveEnd_t = moveStart_t + moveDuration_t,\ t \in allTasks$$

Figure 51: The end time for the move to task t from its predecessor

$$toolUsed_t \in tools,\ t \in allTasks$$

Figure 52: The tool used at task t

$$taskSubComponents_t \subset components,\ t \in tasks$$

Figure 53: The components that form the sub assemblies used in task t. The components can in them selves be sub assemblies

$$taskCompleteSubComponents_t \subset components,\ t \in tasks$$

Figure 54: The components that form the sub assemblies used in task t. This includes all sorts of components, being sub assemblies them selves or *primitive* components.

$$subComponents_c \subset components,\ c \in components$$

Figure 55: The *prmitive* components that form the component c. By being *primitive*, the components in the set cannot be sub assemblies them selves.

$$taskOutOfRange_m \subset tasks,\ m \in machines$$

Figure 56: The tasks that cannot be reached by machine m

Filter

Domain filter

$$maxMoveDurs_t = max(\{timeMatrix3D_{t,j,k} : \forall j \in tasks,\\ \forall k \in \{1 \dots timeMatrixDepth\},\ j \neq t\}),\ \forall t \in tasks$$

Figure 57: The maximum duration for a move to task t

$$minMoveDurs_t = min(\{timeMatrix3D_{t,j,k} : \forall j \in tasks,\\ \forall k \in \{1 \dots timeMatrixDepth\},\ j \neq t\}),\ \forall t \in tasks$$

Figure 58: The minimum duration for a move to task t

$$maxEnd = \sum_{\forall t \in tasks} duration_t + \sum_{\forall t \in tasks} maxMoveDurs_t$$

Figure 59: The upper limit of the schedule; all tasks is laid out after one after another and the duration between them is the maximum of the moves to them

$$minEnd = \left(\sum_{\forall t \in tasks} duration_t + \sum_{\forall t \in tasks} minMoveDurs_t \right) / nbrMachines$$

Figure 60: The lower limit of the schedule; the total duration of each task is the duration of the task itself and the minimum duration of a move to the task, and the tasks are scheduled perfectly over all the machines

$$start_t \leq maxEnd - d_t \wedge end_t \leq maxEnd,\ \forall t \in allTasks$$

Figure 61: Sets the upper limit for the start of each task to be the maximum end minus the duration for the task. Sets the end for each task to be the maximum end

$$end_t \geq d_t + minMoveDurs_t,\ \forall t \in tasks$$

Figure 62: A task can start at its earliest at the time directly after the move to a task, therefore the end of a task can earliest happen after the duration of the task plus the shortest move to it

$$moveStart_t \leq maxEnd - (d_t + minMoveDurs_t),\ \forall t \in tasks$$

Figure 63: A move to a task can start at the latest $maxEnd$ but before the duration of the task and before at least the minimum of the move times to the task

$$moveDuration_t \leq maxMoveDurs_t \ \wedge moveDuration_t \geq minMoveDurs_t,$$
$$\forall t \in tasks$$

Figure 64: The move duration for task t is limited by $maxMoveDurs$ and $minMoveDurs$

$$moveEnd_t \leq maxEnd - d_t \ \wedge moveEnd_t \geq minMoveDurs_t,$$
$$\forall t \in tasks$$

Figure 65: The end of a move to a task can at the latest come at $maxEnd$ minus the duration of the task. The move to a task can at the earliest happen at time 0, so the end can earliest happen at the shortest move time to the task

$$makeSpan \leq maxEnd \wedge makespan \geq minEnd$$

Figure 66: Limits the makespan

$$moveDuration_t \neq i,$$
$$\forall t \in tasks,$$
$$\forall i \in \{0 \ldots maxMoveDurs_t\}/\{timeMatrix3D_{task,j,k} : \forall j \in tasks,$$
$$\forall k \in \{1 \ldots timeMatrixDepth\}, t \neq j\}$$

Figure 67: Limits the $moveDuration$ domains to only the values specified in the $timeMatrix3D$

$$moveStart_t \geq min(\{d_{tt} + minMoveDurs_{tt} : \forall tt \in taking\}), \ \forall t \in tasks/taking$$

Figure 68: As the schedule has to start with a take task, the move to the other tasks can only start as early as after the shortest move to and execution of one of the take tasks

$$start_t \geq max(\{duration_{pt} + minMoveDurs_{pt} : \forall pt \in prevTasks\}),$$
$$nbrMachines \geq |prevTasks|, \ 0 < |prevTasks|,$$
$$prevTasks = \{task : \forall task \in tasks, \ componentCreated_{task} \in componentsUsed_t\},$$
$$\forall t \in tasks$$

Figure 69: $prevTasks$ are the tasks for which the task t uses the component created at task $task$, hence the tasks in $prevTasks$ precedes task t. If the number of machines are greater than or equal to the number of task preceding task t, then the best scheduling is to do all tasks in parallel. If so the earliest task t can start is greater or equal to the maximum of the preceding tasks

$$start_t \geq \left(\sum_{\forall pt \in prevTasks} duration_{pt} + minMoveDurs_{pt} \right) / nbrMachines,$$
$$nbrMachines < |prevTasks|,$$
$$prevTasks = \{task : \forall task \in tasks,\ componentCreated_{task} \in componentsUsed_t\},$$
$$\forall t \in tasks$$

Figure 70: $prevTasks$ are the tasks for which the task t uses the component created at task $task$, hence the tasks in $prevTasks$ precedes task t. If the number of machines are less than the number of tasks preceding task t, then the best we can do is to divide the task times equally on all machines. If the tasks can be divided onto the machines so that the total length of the times on all machines are the same, that time will be equal to the sum$/nbrMachines$. If they don't match up the maximum of these times will be larger than the sum$/nbrMachines$.

$$end_t \leq maxEnd - max(\{duration_{st} + minMoveDurs_{st} : \forall st \in succTasks\}),$$
$$nbrMachines \geq |succTasks|,\ 0 < |succTasks|,$$
$$succTasks = \{task : \forall task \in tasks,\ componentsUsed_t \subset taskCompleteSubComponent_{task},$$
$$componentsUsed_t \cup taskCompleteSubComponents_{task} \neq \emptyset\},$$
$$\forall t \in tasks$$

Figure 71: $succTasks$ are the tasks that has the components used in task t as subcomponents, hence the tasks in $succTasks$ succeeds task t. If the number of machines are greater than or equal to the number of task preceding task t, then the best scheduling is to do all tasks in parallel. If so the latest task t can end is less than or equal to the maximum end of the schedule minus the longest of the succeeding tasks

$$end_t \leq maxEnd - \left(\sum_{\forall st \in succTasks} duration_{st} + minMoveDurs_{st} \right) / nbrMachines,$$
$$nbrMachines \leq |succTasks|,$$
$$succTasks = \{task : \forall task \in tasks,$$
$$componentsUsed_t \subset taskCompleteSubComponent_{task},$$
$$componentsUsed_t \cup taskCompleteSubComponents_{task} \neq \emptyset\},$$
$$\forall t \in tasks$$

Figure 72: $succTasks$ are the tasks that has the components used in task t as subcomponents, hence the tasks in $succTasks$ succeeds task t. If the number of machines are less than the number of tasks preceding task t, then the best we can do is to divide the task times equally on all machines. If the tasks can be divided onto the machines so that the total length of the times on all machines are the same, that time will be equal to the sum/$nbrMachines$. If they don't match up the maximum of these times will be larger than the sum/$nbrMachines$.

Predecessor filter

$$alldifferent(\{pred_t : \forall t \in tasks\})$$

Figure 73: Helps ensure that no two tasks can have the same predecessor

$$pred_{t1} \neq t2,\ t1 \neq t2,\ \forall t1, \forall t2 \in taking$$

Figure 74: No two taking tasks can be the predecessor of each other

$$pred_{t1} \neq t2,\ t1 \neq t2,\ \forall t1, \forall t2 \in putting$$

Figure 75: No two putting tasks can be the predecessor of each other

$$pred_{t1} \neq t2,\ t1 \neq t2,\ \forall t1, \forall t2 \in mounting$$

Figure 76: No two mounting tasks can be the predecessor of each other

$$\begin{aligned}
&pred_t \neq nonPred,\\
&\forall nonPred \in nonPredecessors,\\
&nonPredecessors = \{t2 : \forall t2 \in tasks,\\
&componentsUsed_t \subset taskCompleteSubComponents_{t2} \vee\\
&componentsUsed_t \subset subComponents_{componentCreated_{t2}}\}\\
&\forall t \in tasks
\end{aligned}$$

Figure 77: A task t cannot have task $t2$ as predecessor if task $t2$ uses a component, or creates a component, that the component task t uses has as a subcomponent

$$pred_{putTask} \neq startTask,\ \forall putTask \in putting,\ \forall startTask \in startTasks$$

Figure 78: Since a component has to be taken before it can be put anywhere, put tasks cannot be first in the schedule

$$pred_{putTask} \neq startTask,\ \forall mountTask \in mounting,\ \forall startTask \in startTasks$$

Figure 79: Since a component has to be taken before it can be mounted anywhere, mount tasks cannot be first in the schedule

$$pred_{goalTask} \neq takeTask,\ \forall takeTask \in taking,\ \forall goalTask \in goalTasks$$

Figure 80: Since a schedule has to end with an assembly on the output, a take task cannot be at the end of the assembly

$$\begin{aligned} global\_cardinality(goalPreds,\ outputTasks,\ counts) \wedge \sum counts > 0, \\ goalPreds = \{pred_{task} : \forall task \in goalTasks\}, \\ outputTasks = \{task : \forall task \in tasks,\ output_{task} > 0\}, \\ counts = \{i : \forall task \in outputTasks,\ i \in \{0 \dots 1\}\} \end{aligned}$$

Figure 81: At least one of the output tasks has to be last on one of the circuits

$$\begin{aligned} global\_cardinality(takePreds,\ startTasks,\ counts) \wedge \sum counts > 0, \\ takePreds = \{pred_{task} : \forall task \in taking,\ output_{task} = 0\}, \\ counts = \{i : \forall task \in startTasks,\ i \in \{0 \dots 1\}\} \end{aligned}$$

Figure 82: At least one of the take tasks, thats not on an output, has to be first on one of the circuits

$$\begin{aligned} pred_{putTask} \neq mountTask, \\ \forall putTask \in putting_{comp}, \\ \forall mountTask \in mounting_{comp}, \\ \forall comp \in components \end{aligned}$$

Figure 83: If a set of tasks on a component involves a mount task and a put task, the predecessor of the put task cannot be the mount task

$$\begin{aligned} pred_{takeTask} \neq mountTask, \\ \forall takeTask \in taking_{comp}, \\ \forall mountTask \in mounting_{comp}, \\ \forall comp \in components \end{aligned}$$

Figure 84: If a set of tasks on a component involves a mount task and a take task, the predecessor of the take task cannot be the mount task.

$$
\begin{aligned}
pred_{putTask} \leq takeTask,\\
\forall takeTask \in taking_{comp},\\
tray_{putTask} = tray_{takeTask},\\
\forall putTask \in puttingcomp,\\
tray_{putTask} > 0,\\
\forall comp \in components
\end{aligned}
$$

Figure 85: If a component has a put and take performed on it in a tray, the predecessor of the put task cannot be the take task.

$$
\begin{aligned}
pred_{putTask} \leq takeTask,\\
\forall takeTask \in taking_{comp},\\
fixture_{takeTask} = f \wedge componentsUsed_{putTask} \subset taskSubComponents_{takeTask},\\
\forall putTask \in puttingcomp,\\
fixture_{putTask} = f,\\
\forall f \in fixtures
\end{aligned}
$$

Figure 86: For every put action on a fixture, there is a take action. The predecessor of the put task cannot be the take task.

$$
\begin{aligned}
&pred_{t1} \neq t2 \wedge pred_{t2} \neq t1,\\
&\forall t2 \in concurrentTasks_{group}/\{t1\},\\
&\forall t1 \in concurrentTasks_{group},\\
&\forall group \in \{1 \ldots nbrConcurrentGroups
\end{aligned}
$$

Figure 87: Concurrent tasks cannot be predecessor to each other.

$$
\begin{aligned}
&pred_{t1} \neq t2,\\
&\forall t2 \in tasks,\ componentCreated_{t1} \in compinentUsed_{t2},\\
&\forall t1 \in tasks,\ componentCreated_{t1} > 0
\end{aligned}
$$

Figure 88: Components cannot be used before they are created.

$$
\begin{aligned}
&pred_{precTask} \neq t,\\
&\forall t \in tasks,\ precTask \neq t,\\
&componentsUsed_{precTask} \cup taskCompleteSubComponents_t \neq \emptyset,\\
&componentUsed_{precTask} \cup taskCompleteSubComponent_t \subset taskCompleteSubComponents_t,\\
&\forall precTask \in tasks
\end{aligned}
$$

Figure 89: Task using a component cannot execute before all the tasks having it as subcomponent.

$$pred_{postTask} \neq preTask,$$
$$\forall predTask \in preTasks,$$
$$\forall postTask \in postTasks,$$
$$preTasks = \{preTask : \forall preTask \in tasks,\ componentsUsed_{preTask} \cap concSubComps \neq \emptyset\},$$
$$concSubComps = \bigcup_{\forall i \in concGroup} taskCompleteSubComponents_i,$$
$$concComps = \bigcup_{\forall i \in concGroup} componentsUsed_i,$$
$$\forall concGroup \in concurrentTasks,\ |concGroup| = nbrMachines$$

Figure 90: If there is a set of concurrent tasks on a subset of tasks using as many machines as available, the tasks after the concurrent tasks cannot have the tasks before the concurrent tasks as predecessors.

Constraints

$$end_t \leq makespan,\ \forall t \in tasks$$

Figure 91: All ends has to be lesser than the total end

$$start_t = 0,\ \forall t \in startTasks \cup goalTasks$$

Figure 92: Start and goal tasks are not temporal tasks, i.e. they are timeless. Therefore, their start time is set to 0

$$usingMachine_{nbrTasks+m} = m \wedge usingMachine_{nbrTasks+nbrMachines+m} = m,$$
$$\forall m \in machines$$

Figure 93: The start tasks and goal tasks are assigned to machines, thereby there are start and goal tasks assigned to every machine. Because of the way start and goal tasks are created, the start tasks starts with number $nbrTasks+1$, and the corresponding goal task for a start task can be accessed by $startTask + nbrMachines$.

$$usingMachine_t \neq m,\ \forall t \in tasksOutOfRange_m,\ \forall m \in machines$$

Figure 94: Setting the tasks that are out of range for each machine

Precedences

$$
\begin{aligned}
&end_{putTask} \leq moveStart_{mountTask}, \\
&\forall putTask \in puttingcomp, \\
&\forall mountTask \in mounting_{comp}, \\
&\forall comp \in components
\end{aligned}
$$

Figure 95: If a set of tasks on a component involves a mount task and a put task, the put task has to come before the mount task

$$
\begin{aligned}
&end_{takeTask} \leq moveStart_{mountTask}, \\
&\forall takeTask \in taking_{comp}, \\
&\forall mountTask \in mounting_{comp}, \\
&\forall comp \in components
\end{aligned}
$$

Figure 96: If a set of tasks on a component involves a mount task and a take task, the take task has to come before the mount task

$$
\begin{aligned}
&end_{putTask} \leq moveStart_{takeTask}, \\
&\forall takeTask \in taking_{comp}, \\
&tray_{putTask} = tray_{takeTask}, \\
&\forall putTask \in puttingcomp, \\
&tray_{putTask} > 0, \\
&\forall comp \in components
\end{aligned}
$$

Figure 97: If a component has a put and take performed on it in a tray, the put has to come before the take.

$$
\begin{aligned}
&end_{putTask} \leq moveStart_{takeTask}, \\
&\forall takeTask \in taking_{comp}, \\
&fixture_{takeTask} = f \wedge componentsUsed_{putTask} \subset taskSubComponents_{takeTask}, \\
&\forall putTask \in puttingcomp, \\
&fixture_{putTask} = f, \\
&\forall f \in fixtures
\end{aligned}
$$

Figure 98: For every put action on a fixture, there is a take action. The put action has to come before the take action.

$$cumulative([moveStart_{task} : \forall task \in puts],\ [abs(end_{takes_i} - moveStart_{puts_i}) : \forall i \in \{1 \dots |puts|\}],$$
$$[1 : \forall i \in \{1 \dots |puts|\}], 1),$$
$$takes = [\underset{\forall take \in takesForEachPut_p}{\arg\min} (taskCompleteSubComponent_{take}) : \forall p \in \{1 \dots |puts|\}],$$
$$takesForEachPut = [\{take : \forall take \in taking,\ fixture_{take} = f,$$
$$componentsUsed_{put} \subset taskCompleteSubComponent_{take}\} : \forall put \in puts],$$
$$puts = [put : \forall put \in putting,\ fixture_{put} = f],$$
$$\forall f \in fixtures$$

Figure 99: The intervals between when components are put and then taken again cannot overlap on the same fixture.

$$start_{t1} = start_{t2} \wedge usingMahine_{t1} \neq usingMachine_{t2},$$
$$\forall t2 \in concurrentTasks_{group}/\{t1\},$$
$$\forall t1 \in concurrentTasks_{group},$$
$$\forall group \in \{1 \dots nbrConcurrentGroups$$

Figure 100: Concurrent tasks has to happen at the same time.

$$moveStart_{t2} \geq end_{t1},$$
$$\forall t2 \in tasks,\ componentCreated_{t1} \in compinentUsed_{t2},$$
$$\forall t1 \in tasks,\ componentCreated_{t1} > 0$$

Figure 101: Components cannot be used before they are created.

$$end_{precTask} \leq moveStart_t,$$
$$\forall t \in tasks,\ precTask \neq t,$$
$$componentsUsed_{precTask} \cup taskCompleteSubComponents_t \neq \emptyset,$$
$$componentUsed_{precTask} \cup taskCompleteSubComponent_t \subset taskCompleteSubComponents_t,$$
$$\forall precTask \in tasks$$

Figure 102: Task using a component cannot execute before all the tasks having it as subcomponent.

$$cumulative([start_t : \forall t \in fixtureTasks],\ [duration_t : \forall t \in fixtureTasks],\ [1 : t \in fixtureTasks],\ 1),$$
$$fixtureTasks = [t : \forall t \in tasks,\ fixture_t = f],$$
$$\forall f \in fixtures$$

Figure 103: Tasks on the same fixture cannot overlap.

$cumulative([start_t : \forall t \in trayTasks],\ [duration_t : \forall t \in trayTasks],\ [1 : t \in trayTasks],\ 1),$
$trayTasks = [t : \forall t \in tasks,\ tray_t = tr],$
$\forall tr \in trays$

Figure 104: Tasks on the same tray cannot overlap.

$cumulative([start_t : \forall t \in outputTasks],\ [duration_t : \forall t \in outputTasks],\ [1 : t \in outputTasks],\ 1),$
$outputTasks = [t : \forall t \in tasks,\ output_t = o],$
$\forall o \in outputs$

Figure 105: Tasks on the same output cannot overlap.

$$Start_t \geq moveEnd_t, \forall t \in tasks$$

Figure 106: A task can only start after the move to it.

Predecessors

$$moveStart_t \geq end_{pred_t},\ \forall t \in tasks$$

Figure 107: A task has to start after its predecessor.

$pred_{startTask} = startTask{+}nbrMachines{-}1,\ \forall startTask \in startTasks/\{nbrTasks{+}1\}$

Figure 108: In order to create a circuit containing the sub circuits, for all start tasks, except the first one, the start tasks predecessor is the previous goal task.

$$pred_{nbrTasks+1} = nbrTasks + nbrMachines \times 2$$

Figure 109: To complete the circuit, the first start tasks predecessor is the last goal task.

$$circuit(\{pred_t : \forall t \in tasks\})$$

Figure 110: The predecessors has to form a circuit.

$pred_{mountTask} = takeTask,$
$\forall takeTask \in taking_c,\ takeTask \notin orderedSet,\ puts = \emptyset,$
$puts = \{p : \forall p \in putting_c,\ (fixture_p > 0 \wedge fixture_p = fixture_{mountTask}) \vee$
$(output_p > 0 \wedge output_p = output_{mountTask}) \vee$
$(tray_p > 0 \wedge tray_p = tray_{mountTask})\},$
$\forall mountTask \in mounting_c,$
$\forall c \in components$

Figure 111: If a set of tasks on a component involves a mount and a take task, but no move tasks or put task on the same fixture, tray or output as the mount, the take task is the predecessor of the mount task.

$$
\begin{aligned}
&pred_{putTask} = takeTask, \\
&\forall takeTask \in taking_c, \\
&\forall putTask \in putting_c,\ tray_{putTask} = 0, \\
&\forall c \in components,\ moving_c = \emptyset
\end{aligned}
$$

Figure 112: If a set of tasks on a component involves a put task not in a tray and a take task, and there is no moves involved, the take task has to be the predecessor of the put task.

$$
\begin{aligned}
&pred_{orderedGroup_{k,i+1}} = orderedGroup_{k,i}, \\
&\forall i \in \{1 \dots |orderedGroup_k| - 1\} \\
&\forall k \in \{1 \dots nbrOrderedGroups\}
\end{aligned}
$$

Figure 113: Sets up the predecessors in accordance with the ordered groups.

$$usingMachine_t = usingMachine_{pred_t},\ \forall t \in tasks \cup goalTasks$$

Figure 114: A task has to use the same machine as its predecessor.

$$moveDuration_t = timeMatrix3D_{pred_t,\ t,\ abs(toolUsed_t - toolUsed_{pred_t})+1},\ \forall t \in tasks$$

Figure 115: Take tasks has to use the same tool as its predecessor or do a change first.

$$toolUsed_t = toolNeeded_t,\ \forall t \in tasks,\ toolNeeded \neq 0$$

Figure 116: Set the tool used for each task in accordance with $toolNeeded$.