Model

Beskriv grunder i hur modellen fungerar There are moves put inbetween each task Model deosn't deal with coordinates only times for movement between tasks
Terminology:
The goal for the assembly is to assemble components. This is components fed to the assembly from the outside, for example button components. The final assembly is the complete assembly of components that make the final product. All the smaller assemblies before that are called sub-assemblies. For reasons explained further down, we will in this thesis call components such as buttons for *pure* components instead of just components.

Variables

Static variables

Static variables are variables that have a fixed value, or is a set or list containing fixed values.

$$nbrTasks \in \{1, \ldots, 2^{32} - 1\} \tag{1}$$

$$tasks = \{1, \ldots, nbrTasks\} \tag{2}$$

First we define the number of tasks to be scheduled. Each task is identified

As mentioned, this model is based on the technique of using predecessors to determine which task comes directly before another. This creates the need to have source and a sink node for each machine, we call them start tasks and goal tasks. As they are not provided as parameters, the model creates them and give them identifiers with numbers above the tasks to be scheduled. As seen later, this is a convenient way of numbering these tasks.

$$startTasks = \{nbrTasks + 1, \ldots, nbrTasks + nbrMachines\} \tag{3}$$

A sources, the start tasks are the predecessors to the first tasks for each machine.

$$goalTasks = \{nbrTasks + nbrMachines + 1, \ldots, nbrTasks + nbrMachines \times 2\} \tag{4}$$

As sinks, the goal tasks has the last task as predecessor for each arm.

$$allTasks = tasks \cup startTasks \cup goalTasks \tag{5}$$

All the tasks, including the start and goal tasks

$$nbrMachines \in \{1, \ldots, 2^{32} - 1\} \tag{6}$$

Then the number of machines available for the assembly, $nbrMachines$. A machine in our case is an arm.

$$machines = \{1, \ldots, nbrMachines\} \tag{7}$$

The machines.

$$nbrTools \in \{1, \ldots, 2^{32} - 1\} \tag{8}$$

The number of tools available, $nbrTools$. These are the number of tools that can be fitted on an arm. The model assumes that there is a set of $nbrTools$ for each machine. I.e. if $nbrTools = 2$ and $nbrMachines = 2$, there is a set of tool 1 and tool 2 for machine 1, and another set of tools 1 and 2 for machine 2. There cannot be a combination of tools such as, for example, only tool 1 for machine 1 and a set of tools 1 and 2 for machine 2.

$$tools = \{1, \ldots, nbrTools\} \tag{9}$$

The tools

$$toolNeeded(t) \in tools, \ t \in tasks \tag{10}$$

The tool needed for task t

$$nbrComponents \in \{1, \ldots, 2^{32} - 1\} \tag{11}$$

$nbrComponents$ defines the number of components used. All components needs to be uniquely identified in the assembly, so even if we use 4 screws in an assembly, we need to define all 4 screws. As mentioned before we distinguish between components and *pure* components. The reason for that is that in the model we do not distinguish between a *pure* component and a sub-assembly, they are the same. And in the model we call them components. The reason for this is because we found it easier to only have one sort of object to deal with when it comes to what will be assembled, instead of two. This means that the final assembly is also a component, i.e. the product produced by the assembly is a component. In other words, in this thesis *pure* components and sub-assemblies are sub sets of components.

$$components = \{1, \ldots, nbrComponents\} \tag{12}$$

The components.

$$taskSubComponents(t) \subset components, \ t \in tasks \tag{13}$$

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

$$taskCompleteSubComponents(t) \subset components, \ t \in tasks \tag{14}$$

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.

$$subComponents(c) \subset components, \ c \in components \tag{15}$$

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

$$nbrTrays \in \{1, \ldots, 2^{32} - 1\} \tag{16}$$

The number of trays available in the assembly, $nbrTrays$. Trays are used to hold components until we need them in the assembly. This can be that the tray holds the components from the beginning, as with *pure* components fed to the assembly, or it can be a sub-assembly put there during the assembly to be picked up again later. Each *pure* component has its own tray, so we can have a button tray, a cover tray, etc.

$$tray(t) \in trays \cup \{0\}, \ t \in tasks \tag{17}$$

$tray(t)$ is the tray task t uses. If no tray is used by the task, $tray(t) = 0$. A tray can either hold a component from the beginning or have a component put in it during the assembly.

$$trays = \{1, \ldots, nbrTrays\} \tag{18}$$

The trays.

$$nbrFixtures \in \{1, \ldots, 2^{32} - 1\} \tag{19}$$

$nbrFixtures$ defines the number of fixtures available in the assembly. A fixture is primarily used to hold a component in order for another component to be mounted on that component. Although, as will be shown in the assembly example [section?], the fixture can be used for purposes than just holding components.

$$fixture(t) \in fixtures \cup \{0\}, \ t \in tasks \tag{20}$$

$fixture(t)$ is the fixture task t uses. If no fixture is used by the task, $fixture(t) = 0$

$$fixtures = \{1, \ldots, nbrFixtures\} \tag{21}$$

The fixtures.

$$nbrOutputs \in \{1, \ldots, 2^{32} - 1\} \tag{22}$$

$nbrOutputs$ defines the number of outputs available. An output is the final stage for a component in an assembly. After it is put here, it will not be removed. Although, there can still be other components mounted on the component put

on the output. In that respect an output can be viewed as a fixture, only that the components put there can not be removed.

$$output(t) \in outputs \cup \{0\}, \ t \in tasks \tag{23}$$

$output(t)$ is the output used by task t. If no output is used by the task, $output(t) = 0$.

$$outputs = \{1, \ldots, nbrOutputs\} \tag{24}$$

The outputs.

$$nbrConcurrentGroups \in \{1, \ldots, 2^{32} - 1\} \tag{25}$$

$nbrConcurrentGroups$ defines the number of concurrent groups used. A concurrent group is a group of tasks that has to be performed at the same time. Hence, a concurrent group can not be larger than the amount of machines available, although, there is no check for it in the model.

$$concurrentGroups = \{1, \ldots, nbrConcurrentGroups\} \tag{26}$$

The concurrent groups.

$$nbrOrderedGroups \in \{1, \ldots, 2^{32} - 1\} \tag{27}$$

The number of ordered groups used. An ordered group is an array of tasks that has to come in a very specific order. An example of this could be if an assembly has many move tasks that needs to be performed one after another in order to make an intricate movement. As will be showed in the constraints[section?], we can reason the relation between tasks if they use a certain component and are a certain kind of action. But we can not reason using two move tasks, there is no way to tell which should come before the other based on the component they use.

$$orderedGroup(k) \subset tasks, \ k \in \{1, \ldots, nbrOrderedGroups\} \tag{28}$$

Ordered group number k. These tasks come in the order in which they should be scheduled. The first one is the predecessor of the second one, the second one is the predecessor of the third one, and so on.

$$order(k) \subset tasks, \ k \in orderedGroups \tag{29}$$

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

$$order(k,i) \in tasks, \ i \in \{1, \ldots, |orderedGroup(k)|\}, \ k \in orderedGroups \tag{30}$$

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

$$orderedSet = \bigcup_{\forall k \in orderedGroups} order(k), \ orderedSet \subset tasks \tag{31}$$

The set of tasks member of ordered groups

All the variables described that starts with nbr describes an amount of something, for example $nbrTasks$ describes the number of tasks to schedule. In *Calculated Static Variables* one can see that these values help define sets of numbers. These sets represents the set of those things, for example a set of tasks. So in the case of tasks, each task is represented by a number from 1 to $nbrTasks$. The numbers in one of those sets represents one of the items in the set.

These sets are used to define the following variables, but they are not present in the parameters to the model. They are therefore not included here, but first in *Calculated Static Variables*.

$tray(t)$, $output(t)$ and $fixture(t)$ can not be set at the same time for a task, since that would mean the task is performed at two locations at the same time, although this is not checked by the model. The only restriction for what kind of tasks can be performed using these are that output can not be used by a take task and tray can not be used by a mount task. If ones assembly contains these combinations, the output or tray should be changed to a fixture.

$$componentsUsed(t) \subset components, \ t \in tasks \tag{32}$$

$componentsUsed(t)$ defines the set of components task t uses. A task usually only uses one component at a time, but uses two in the case of mounting tasks, the mounted component and the component mounted on.

$$mounting \subset tasks \tag{33}$$

$$taking \subset tasks \tag{34}$$

$$moving \subset tasks \tag{35}$$

$$putting \subset tasks \tag{36}$$

Each task performed can be classified as either a mount task, a take task, a move task or a put task, but only one of them.

Taking A task that picks up a component is a taking task. The location of the component is specified by either a tray or a fixture, but not an output since there is no reason to pick up something that has been placed on an output.

Mounting A task that mounts a component on another component is a mounting task. This assumes that the component to mount is picked up and in the hand. The location of the component to mount on is defined by either a fixture or an output.

Putting A task that puts a component somewhere is a putting task. Where a component is put is defined by either a fixture, a tray or an output.

Moving A task that moves a components from one place to another is a moving task. The model already puts in moves between tasks and if, for example, the first task is a take task and the second task is a put task, the move in between them is essentially a move that moves a component from one

place to the another. Although, sometimes it can be handy to define a task that explicitly moves a component. An example of that can be if one wants to spin a component around. Then one can specify a take task in order to pick up the component, a move task to turn it, and a put task to put the component back. In this case there will be three moves of the component; one to move from the take task to the move task, the move task itself, and a move from the move task to the put task.

$$putting(c) \subset putting, \; c \in components \tag{37}$$

Set of tasks that puts component c somewhere

$$mounting(c) \subset mounting, \; c \in components \tag{38}$$

Set of tasks that mounts component c

$$taking(c) \subset taking, \; c \in components \tag{39}$$

Set of tasks that takes component c

$$moving(c) \subset moving, \; c \in components \tag{40}$$

Set of tasks that moves component c somewhere

$$concurrentTasks(k) \subset tasks, \; k \in concurrentGroups \tag{41}$$

The k set of tasks needing concurrent execution

$$changeToolDuration(tool_1, tool_2) \in \{0, \ldots, 2^{32} - 1\}, \; tool_1, tool_2 \in tools \tag{42}$$

The duration of changing from $tool_1$ to $tool_2$

$$duration(t) \in \{0, \ldots, 2^{32} - 1\}, \; t \in tasks \tag{43}$$

The duration of task t

$$timeMatrixDepth = \frac{n^2 - n + 2}{2}, \; n = nbrMachines \tag{44}$$

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

$$timeMatrix3D(t(from), t(to), k) \in \{0, \ldots, 2^{32} - 1\}, \; t(from) \in tasks \cup startTasks,$$
$$t(to) \in tasks, \; k \in \{0, \ldots, timeMatrixDepth\} \tag{45}$$

The time to move from task $t(from)$ to task $t(to)$ changing tool according to k

$$taskOutOfRange(m) \subset tasks, \; m \in machines \tag{46}$$

The tasks that cannot be reached by machine m

Decision variables

$$usingMachine(t) \in machines,\ t \in tasks \tag{47}$$

The machine task t uses

$$pred(t) \in allTasks,\ t \in allTasks \tag{48}$$

The predecessor task of task t

$$
\begin{aligned}
maxE = &(max(\{duration(t) : t \in tasks\}) + \\
&max(\{timeMatrix3D(t_1, t_2, k) : \forall t_1 \in tasks \cup startTasks, \\
&\forall t_2 \in tasks,\ \forall k \in \{0, \ldots, timeMatrixDepth\}\}) \times nbrTasks
\end{aligned} \tag{49}
$$

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.

$$start(t) \in \{0, \ldots, maxE\},\ t \in allTasks \tag{50}$$

The start time for task t

$$end(t) = start(t) + duration(t),\ t \in allTasks \tag{51}$$

The end time for task t

$$makespan \in \{0, \ldots, maxE\} \tag{52}$$

The makespan for the whole schedule, the time to minimize

$$moveDuration(t) \in \{0, \ldots, maxE\},\ t \in allTasks \tag{53}$$

The duration of the move to task t from its predecessor

$$moveStart(t) \in \{0, \ldots, maxE\},\ t \in allTasks \tag{54}$$

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

$$moveEnd(t) = moveStart(t) + moveDuration(t),\ t \in allTasks \tag{55}$$

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

$$toolUsed(t) \in tools,\ t \in allTasks \tag{56}$$

The tool used at task t

7

Filter

Domain filter

$$(\forall t \in tasks)$$
$$maxMoveDurs(t) = max(\{timeMatrix3D(t, j, k) :$$
$$\forall j \in tasks, \tag{57}$$
$$\forall k \in \{1, \ldots, timeMatrixDepth\},$$
$$j \neq t\})$$

The maximum duration for a move to task t

$$(\forall t \in tasks)$$
$$minMoveDurs(t) = min(\{timeMatrix3D(t, j, k) :$$
$$\forall j \in tasks, \tag{58}$$
$$\forall k \in \{1, \ldots, timeMatrixDepth\},$$
$$j \neq t\})$$

The minimum duration for a move to task t

$$maxEnd = \sum_{\forall t \in tasks} duration(t) + \sum_{\forall t \in tasks} maxMoveDurs(t) \tag{59}$$

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

$$minEnd = \frac{\left(\sum_{\forall t \in tasks} duration(t) + \sum_{\forall t \in tasks} minMoveDurs(t)\right)}{nbrMachines} \tag{60}$$

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

$$(\forall t \in allTasks)$$
$$start(t) \leq maxEnd - duration(t) \wedge \tag{61}$$
$$end(t) \leq maxEnd$$

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

$$(\forall t \in tasks) \, end(t) \geq duration(t) + minMoveDurs(t) \tag{62}$$

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

$$(\forall t \in tasks) \, moveStart(t) \leq maxEnd - (duration(t) + minMoveDurs(t) \tag{63}$$

8

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

$$
\begin{aligned}
&(\forall t \in tasks) \\
&moveDuration(t) \leq maxMoveDurs(t) \, \wedge \\
&moveDuration(t) \geq minMoveDurs(t)
\end{aligned}
\tag{64}
$$

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

$$
\begin{aligned}
&(\forall t \in tasks) \\
&moveEnd(t) \leq maxEnd - duration(t) \, \wedge \\
&moveEnd(t) \geq minMoveDurs(t)
\end{aligned}
\tag{65}
$$

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

$$
\begin{aligned}
&makeSpan \leq maxEnd \wedge \\
&makespan \geq minEnd
\end{aligned}
\tag{66}
$$

Limits the makespan

$$
\begin{aligned}
&(\forall t \in tasks) \\
&(\forall i \in \{0, \ldots, maxMoveDurs(t)\}/\{timeMatrix3D(task, j, k) : \\
&\qquad\qquad\qquad\qquad\qquad \forall j \in tasks, \\
&\qquad\qquad\qquad\qquad\qquad \forall k \in \{1, \ldots, timeMatrixDepth\} \\
&\qquad\qquad\qquad\qquad\qquad t \neq j\}) \\
&moveDuration(t) \neq i,
\end{aligned}
\tag{67}
$$

Limits the $moveDuration$ domains to only the values specified in the $timeMatrix3D$

$$
\begin{aligned}
&(\forall t \in tasks/taking) \\
&moveStart(t) \geq min(\{duration(tt) + minMoveDurs(tt) : \\
&\qquad\qquad\qquad \forall tt \in taking\})
\end{aligned}
\tag{68}
$$

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

$$
\begin{aligned}
&(\forall t \in tasks) \\
&prevTasks = \{task : \forall task \in tasks, \\
&\qquad\qquad\qquad componentCreated(task) \in componentsUsed(t)\}, \\
&nbrMachines \geq |prevTasks|, \\
&\qquad\quad 0 < |prevTasks|, \\
&start(t) \geq max(\{duration(pt) + minMoveDurs(pt) : \forall pt \in prevTasks\})
\end{aligned}
\tag{69}
$$

$prevTasks$ 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

$$
\begin{aligned}
&(\forall t \in tasks)\\
&prevTasks = \{task : \forall task \in tasks,\\
&\qquad\qquad\qquad componentCreated(task) \in componentsUsed(t)\},\\
&nbrMachines < |prevTasks|,\\
&start(t) \geq \frac{\left(\sum_{\forall pt \in prevTasks} duration(pt) + minMoveDurs(pt)\right)}{nbrMachines}
\end{aligned}
\tag{70}
$$

$prevTasks$ 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/$nbrMachines$.

$$
\begin{aligned}
&(\forall t \in tasks)\\
&succTasks = \{task : \forall task \in tasks,\\
&componentsUsed(t) \subset taskCompleteSubComponent(task),\\
&componentsUsed(t) \cup taskCompleteSubComponents(task) \neq \emptyset\},\\
&nbrMachines \geq |succTasks|,\\
&\qquad\qquad 0 < |succTasks|,\\
&end(t) \leq maxEnd - max(\{duration(st) + minMoveDurs(st) :\\
&\qquad\qquad\qquad \forall st \in succTasks\})
\end{aligned}
\tag{71}
$$

$succTasks$ 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

$$
\begin{aligned}
&(\forall t \in tasks)\\
&succTasks = \{task : \forall task \in tasks,\\
&componentsUsed(t) \subset taskCompleteSubComponent(task),\\
&componentsUsed(t) \cup taskCompleteSubComponents(task) \neq \emptyset\},\\
&nbrMachines \leq |succTasks|,\\
&end(t) \leq maxEnd - \frac{\left(\sum_{\forall st \in succTasks} duration(st) + minMoveDurs(st)\right)}{nbrMachines}
\end{aligned}
$$

$$(72)$$

$succTasks$ 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

$$alldifferent(\{pred(t) : \forall t \in tasks\}) \qquad (73)$$

Helps ensure that no two tasks can have the same predecessor

$$(\forall t1, \forall t2 \in taking)\ pred(t1) \neq t2 \qquad (74)$$

No two taking tasks can be the predecessor of each other

$$(\forall t1, \forall t2 \in taking)\ pred(t1) \neq t2 \qquad (75)$$

No two putting tasks can be the predecessor of each other

$$(\forall t1, \forall t2 \in mounting)\ pred(t1) \neq t2 \qquad (76)$$

No two mounting tasks can be the predecessor of each other

$$
\begin{aligned}
&(\forall t \in tasks) \\
&nonPredecessors = \{t_2 : \forall t_2 \in tasks, \\
&componentsUsed(t) \subset taskCompleteSubComponents(t_2)\ \vee \\
&componentsUsed(t) \subset subComponents(componentCreated(t_2))\} \\
&(\forall nonPred \in nonPredecessors) \\
&pred(t) \neq nonPred,
\end{aligned}
\qquad (77)
$$

A task t cannot have task t_2 as predecessor if task t_2 uses a component, or creates a component, that the component task t uses has as a subcomponent

$$
\begin{aligned}
&(\forall startTask \in startTasks) \\
&(\forall putTask \in putting) \\
&pred(putTask) \neq startTask
\end{aligned}
\qquad (78)
$$

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

$$
\begin{aligned}
&(\forall startTask \in startTasks) \\
&(\forall mountTask \in mounting) \\
&pred(putTask) \neq startTask
\end{aligned}
\qquad (79)
$$

11

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

$$(\forall goalTask \in goalTasks)$$
$$(\forall takeTask \in taking) \tag{80}$$
$$pred(goalTask) \neq takeTask$$

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

$$counts = \{i : \forall task \in outputTasks, \ i \in \{0, \ldots, 1\}\},$$
$$outputTasks = \{task : \forall task \in tasks, \ output(task) > 0\},$$
$$goalPreds = \{pred(task) : \forall task \in goalTasks\}, \tag{81}$$
$$global\_cardinality(goalPreds, \ outputTasks, \ counts) \wedge$$
$$\sum counts > 0$$

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

$$counts = \{i : \forall task \in startTasks, \ i \in \{0, \ldots, 1\}\},$$
$$takePreds = \{pred(task) : \forall task \in taking, \ output(task) = 0\},$$
$$global\_cardinality(takePreds, \ startTasks, \ counts) \wedge \tag{82}$$
$$\sum counts > 0$$

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

$$(\forall comp \in components)$$
$$(\forall mountTask \in mounting(comp))$$
$$(\forall putTask \in putting(comp)) \tag{83}$$
$$pred(putTask) \neq mountTask$$

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

$$(\forall comp \in components)$$
$$(\forall mountTask \in mounting(comp))$$
$$(\forall takeTask \in taking(comp)) \tag{84}$$
$$pred(takeTask) \neq mountTask$$

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.

$$(\forall comp \in components)$$
$$(\forall putTask \in puttingcomp, \ tray(putTask) > 0)$$
$$(\forall takeTask \in taking(comp), \ tray(putTask) = tray(takeTask)) \tag{85}$$
$$pred(putTask) \leq takeTask$$

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.

$$(\forall f \in fixtures)$$
$$(\forall putTask \in puttingcomp, \; fixture(putTask) = f)$$
$$(\forall takeTask \in taking(comp), \; fixture(takeTask) = f,$$
$$componentsUsed(putTask) \subset taskSubComponents(takeTask))$$
$$pred(putTask) \leq takeTask$$

$$(86)$$

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

$$(\forall group \in \{1, \ldots, nbrConcurrentGroups\})$$
$$(\forall t_1 \in concurrentTasks(group))$$
$$(\forall t_2 \in concurrentTasks(group)/\{t_1\})$$
$$pred(t_1) \neq t_2 \wedge pred(t_2) \neq t_1$$

$$(87)$$

Concurrent tasks cannot be predecessor to each other.

$$(\forall t_1 \in tasks, \; componentCreated(t_1) > 0)$$
$$(\forall t_2 \in tasks, \; componentCreated(t_1) \in compinentUsed(t_2))$$
$$pred(t_1) \neq t_2$$

$$(88)$$

Components cannot be used before they are created.

$$(\forall precTask \in tasks)$$
$$(\forall t \in tasks, \; precTask \neq t,$$
$$componentUsed(precTask) \cup taskCompleteSubComponent(t) \subset$$
$$taskCompleteSubComponents(t),$$
$$componentsUsed(precTask) \cup taskCompleteSubComponents(t) \neq \emptyset$$
$$pred(precTask) \neq t$$

$$(89)$$

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

$$(\forall concGroup \in concurrentTasks, \; |concGroup| = nbrMachines)$$

$$concComps = \bigcup_{\forall i \in concGroup} componentsUsed(i),$$

$$concSubComps = \bigcup_{\forall i \in concGroup} taskCompleteSubComponents(i),$$

$$preTasks = \{preTask : \forall preTask \in tasks,$$

$$componentsUsed(preTask) \cap concSubComps \neq \emptyset\},$$

$$(\forall postTask \in postTasks)$$

$$(\forall predTask \in preTasks)$$

$$pred(postTask) \neq preTask$$

$$(90)$$

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

$$(\forall t \in tasks) \; end(t) \leq makespan \tag{91}$$

All ends has to be lesser than the total end

$$(\forall t \in startTasks \cup goalTasks) \; start(t) = 0 \tag{92}$$

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

$$(\forall m \in machines)$$

$$usingMachine(nbrTasks + m) = m \; \wedge \tag{93}$$

$$usingMachine(nbrTasks + nbrMachines + m) = m$$

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 $startTask + nbrMachines$.

$$(\forall m \in machines)$$

$$(\forall t \in tasksOutOfRange(m)) \tag{94}$$

$$usingMachine(t) \neq m$$

Setting the tasks that are out of range for each machine

Precedences

$$(\forall comp \in components)$$
$$(\forall mountTask \in mounting(comp))$$
$$(\forall putTask \in putting(comp))$$
$$end(putTask) \leq moveStart(mountTask) \tag{95}$$

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

$$\forall comp \in components$$
$$\forall mountTask \in mounting(comp),$$
$$\forall takeTask \in taking(comp),$$
$$end(takeTask) \leq moveStart(mountTask) \tag{96}$$

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

$$\forall comp \in components$$
$$(\forall putTask \in putting(comp),\ tray(putTask) > 0)$$
$$(\forall takeTask \in taking(comp), tray(putTask) = tray(takeTask))$$
$$end(putTask) \leq moveStart(takeTask) \tag{97}$$

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

$$(\forall f \in fixtures)$$
$$(\forall putTask \in putting(comp),\ fixture(putTask) = f)$$
$$(\forall takeTask \in taking(comp),\ fixture(takeTask) = f\ \land$$
$$componentsUsed(putTask) \subset taskSubComponents(takeTask))$$
$$end(putTask) \leq moveStart(takeTask),$$

$$\tag{98}$$

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

$$
\begin{aligned}
&(\forall f \in fixtures) \\
&puts = [put : \forall put \in putting, \ fixture(put) = f], \\
&takesForEachPut = [\{take : \forall take \in taking, \ fixture(take) = f, \\
&componentsUsed(put) \subset taskCompleteSubComponent(take)\} : \forall put \in puts], \\
&takes = [\underset{\forall take \in takesForEachPut(p)}{\arg\min} \ taskCompleteSubComponent(take) : \\
&\qquad \forall p \in \{1, \ldots, |puts|\}], \\
&cumulative([moveStart(task) : \forall task \in puts], \\
&\qquad [abs(end(takes(i)) - moveStart(puts(i))) : \forall i \in \{1, \ldots, |puts|\}], \\
&\qquad [1 : \forall i \in \{1, \ldots, |puts|\}], \\
&\qquad 1)
\end{aligned}
\tag{99}
$$

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

$$
\begin{aligned}
&(\forall group \in \{1, \ldots, nbrConcurrentGroups\}) \\
&\quad (\forall t_1 \in concurrentTasks(group)) \\
&\quad (\forall t_2 \in concurrentTasks(group)/\{t_1\}) \\
&start(t_1) = start(t_2) \ \wedge \\
&usingMahine(t_1) \neq usingMachine(t_2),
\end{aligned}
\tag{100}
$$

Concurrent tasks has to happen at the same time.

$$
\begin{aligned}
&(\forall t_1 \in tasks, \ componentCreated(t1) > 0) \\
&\quad (\forall t_2 \in tasks, \ componentCreated(t_1) \in compinentUsed(t_2)) \\
&moveStart(t_2) \geq end(t_1)
\end{aligned}
\tag{101}
$$

Components cannot be used before they are created.

$$
\begin{aligned}
&(\forall precTask \in tasks) \\
&\quad (\forall t \in tasks, \ precTask \neq t, \\
&\qquad componentUsed(precTask) \cup taskCompleteSubComponent(t) \subset \\
&\qquad taskCompleteSubComponents(t), \\
&\qquad componentsUsed(precTask) \cup taskCompleteSubComponents(t) \neq \emptyset) \\
&end(precTask) \leq moveStart(t),
\end{aligned}
\tag{102}
$$

Task using a component cannot execute before all the tasks having it as sub-component.

$$(\forall f \in fixtures)$$
$$fixtureTasks = [t : \forall t \in tasks, \ fixture(t) = f],$$
$$cumulative([start(t) : \forall t \in fixtureTasks],$$
$$[duration(t) : \forall t \in fixtureTasks],$$
$$[1 : t \in fixtureTasks],$$
$$1) \tag{103}$$

Tasks on the same fixture cannot overlap.

$$(\forall tr \in trays)$$
$$trayTasks = [t : \forall t \in tasks, \ tray(t) = tr],$$
$$cumulative([start(t) : \forall t \in trayTasks],$$
$$[duration(t) : \forall t \in trayTasks],$$
$$[1 : t \in trayTasks],$$
$$1) \tag{104}$$

Tasks on the same tray cannot overlap.

$$(\forall o \in outputs)$$
$$outputTasks = [t : \forall t \in tasks, \ output(t) = o],$$
$$cumulative([start(t) : \forall t \in outputTasks],$$
$$[duration(t) : \forall t \in outputTasks],$$
$$[1 : t \in outputTasks],$$
$$1) \tag{105}$$

Tasks on the same output cannot overlap.

$$(\forall t \in tasks) \ Start(t) \geq moveEnd(t) \tag{106}$$

A task can only start after the move to it.

Predecessors

$$(\forall t \in tasks) \ moveStart(t) \geq end(pred(t)) \tag{107}$$

A task has to start after its predecessor.

$$(\forall startTask \in startTasks/\{nbrTasks + 1\})$$
$$pred(startTask) = startTask + nbrMachines - 1 \tag{108}$$

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.

$$pred(nbrTasks + 1) = nbrTasks + nbrMachines \times 2 \tag{109}$$

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

$$circuit(\{pred(t) : \forall t \in tasks\}) \tag{110}$$

The predecessors has to form a circuit.

$$(\forall c \in components)$$
$$(\forall mountTask \in mounting(c))$$
$$puts = \{p : \forall p \in putting(c),$$
$$(fixture(p) > 0 \wedge fixture(p) = fixture(mountTask)) \vee$$
$$(output(p) > 0 \wedge output(p) = output(mountTask)) \vee$$
$$(tray(p) > 0 \wedge tray(p) = tray(mountTask))\},$$
$$(\forall takeTask \in taking(c), \ takeTask \notin orderedSet, \ puts = \emptyset)$$
$$pred(mountTask) = takeTask$$
$$\tag{111}$$

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.

$$(\forall c \in components, \ moving(c) = \emptyset)$$
$$(\forall putTask \in putting(c), \ tray(putTask) = 0)$$
$$(\forall takeTask \in taking(c)) \tag{112}$$
$$pred(putTask) = takeTask$$

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.

$$(\forall k \in \{1, \ldots, nbrOrderedGroups\})$$
$$(\forall i \in \{1, \ldots, |orderedGroup(k)| - 1\}) \tag{113}$$
$$pred(orderedGroup(k, i + 1)) = orderedGroup(k, i)$$

Sets up the predecessors in accordance with the ordered groups.

$$(\forall t \in tasks \cup goalTasks)$$
$$usingMachine(t) = usingMachine(pred(t)) \tag{114}$$

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

$$(\forall t \in tasks)$$

$$k = abs(toolUsed(t) - toolUsed(pred(t))) + 1, \qquad (115)$$

$$moveDuration(t) = timeMatrix3D(pred(t),\ t,\ k)$$

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

$$(\forall t \in tasks,\ toolNeeded(t) \neq 0)\ toolUsed(t) = toolNeeded(t) \qquad (116)$$

Set the tool used for each task in accordance with *toolNeeded*.