<http://www.diva-portal.org>

Postprint

This is the accepted version of a paper presented at *19th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'14), Barcelona, Spain, 16-16 September, 2014.*

Citation for the original published paper:

Akan, B., Ameri E., A., Curuklu, B. (2014)

Scheduling for Multiple Type Objects Using POPStar Planner.

In: *Proceedings of the 19th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'14), Barcelona, Spain, September, 2014* (pp. 1-7).

N.B. When citing this work, cite the original published paper.

Permanent link to this version:

<http://urn.kb.se/resolve?urn=urn:nbn:se:mdh:diva-26465>

Scheduling for Multiple Type Objects Using POPStar Planner

Batu Akan, Afshin Ameri E., Baran Çürüklü

batu.akan@mdh.se, afshin.ameri@mdh.se, baran.curuklu@mdh.se

Mälardalen University, Västerås, Sweden

Abstract

In this paper, scheduling of robot cells that produce multiple object types in low volumes are considered. The challenge is to maximize the number of objects produced in a given time window as well as to adopt the schedule for changing object types. Proposed algorithm, POPStar, is based on a partial order planner which is guided by best-first search algorithm and landmarks. The best-first search, uses heuristics to help the planner to create complete plans while minimizing the makespan. The algorithm takes landmarks, which are extracted from user's instructions given in structured English as input. Using different topologies for the landmark graphs, we show that it is possible to create schedules for changing object types, which will be processed in different stages in the robot cell. Results show that the POPStar algorithm can create and adapt schedules for robot cells with changing product types in low volume production.

1 Introduction

Nowadays, industrial robots are an integrated part of the manufacturing industry. These tools have been around since the eighties and their role in increasing productivity as well as quality improvement is well documented [6]. Still, from the customer viewpoint, few industries dominate the industrial robot landscape. The ever competitive automotive industry being the dominant one [14].

The rationale applied to large industries, e.g. the automotive industry, is radically different from that of small and medium size enterprises (SMEs). Still, the SME sector aims to address roughly the same challenges, primarily, increasing productivity and quality improvement. Diffusion of industrial robots in the SME sector is, however, beyond that of their larger counterparts. We have earlier addressed the rationale behind this difference [1]. In the core of our hypothesis is the notion that (re)programming an industrial robot requires expert knowledge, and this knowledge is rarely found in-house at SMEs. For the majority, this implies that they need to depend on external assistance that is associated with high costs and in the worst case, unacceptable time delays.

Typical use of an industrial robot is in machine tend-

ing, a process in which a robot loads/unloads material into machines that are located in a robot cell. Unprocessed parts¹ are picked up from an input palette or a conveyor band. Similar to the classical flow shop scheduling problem, parts are successively processed through the machines. The challenge in machine tending applications is to maximize number of produced products in a given time window. As the number of machines in the robot cell increases, scheduling the movements of the robot gets more complex. Additional challenges can be introduced to this problem by adding parallel machines, robots with dual grippers [10] and different object types produced simultaneously in the same cell.

The approach for programming an industrial robot proposed in this paper is task-level programming. The user interacts with an industrial robot by giving instructions in structured English which are processed as described in [2]. The instructions represent the tasks that need to be performed by the robot in order to solve a given problem. Inherent characteristics of natural language imply that user instructions might be incomplete. In other cases, entire chains of actions may be missing. The missing information in the user's instructions should be filled by a planner. The planner must also choose appropriate actions such that the overall plan has the minimal makespan. This is highly beneficial for an SME who has low volume productions and needs to constantly reconfigure the cell for different types of products without the need of an integrator.

In this paper, realistic low volume production cases are handled using the POPStar algorithm. For the integrator the algorithm can generate skeleton code to develop onto and for the SME it provides means to reconfigure the cell for manufacturing parts that requires different types of processing stages. The algorithm creates filling and emptying sequences for the cell, as well as optimum cycles for operating the machinery and the robot. This paper evaluates POPStar in robot cells that produce multiple types of objects. This case means that these multiple object types pass through different sequences of machines in the robot cell. Consequently the POPStar algorithm can also handle cases in which, one object type will be processed after the plan associated with the previous object type is finished.

¹The words *part* and *object* are used interchangeably through out the text depending on the context.

The remainder of this paper is organized as follows: Section 2 presents our assumptions of the robot scheduling problem. Also a review of existing work on robot scheduling problems and symbolic planning are given. A short review of the partial planning algorithm and the modifications that are proposed in order to solve the given planning problem are presented in Section 3. Section 4 evaluates our approach with sample test cases. Finally, the conclusions are presented in Section 5.

2 Background

A typical robot cell is a flow shop scheduling problem [9]. It consists of m processing stages, each with one or more machines, an input device I , which stores the raw/unprocessed parts, an output device O , to leave the finished product, and one or more robots that transfer the parts between the stages. Each unprocessed part needs to go through all the stages in a fixed order. The number of machines in each stage, number of grippers on each robot and the number of robots, specify the robot cell problem. A very detailed classification of types of robot cell scheduling problems is given in [5].

Following the notation described in [10] let $M = 1, 2, \dots, m$ be the set of indices for the m processing stages of a robot cell. Each stage s has $m_s \geq 1$ identical parallel machines. The machines in stage s are denoted $M_{s_1}, M_{s_2}, \dots, M_{sm_s}$. Each machine in stage s requires time p_s to process a part.

Robot scheduling problems have received a growing interest over the past few decades from the scheduling research community. Levner *et al.* [13] present survey of complexity analysis of cyclic scheduling problems.

Particularly, robot cell scheduling problems, where travel times between machines are not negligible, have received fair amount of interest by the practitioners of industry. After all, in a production facility, the throughput of a robot cell is highly dependent on the robot's movements between the machines. A very detailed classification of types of robot cell scheduling problems is given in [5].

In their work, Fan and Winley [7] use a heuristic search algorithm for the flow shop scheduling problem. Their approach employs best-first search. Even though the work presented in their paper is not for a robot cell, their heuristic guided approach can be applied to robot cell scheduling problem as well.

Other solutions for the general problem include mathematical methods such as mixed integer programming. In their work Koné *et al.* [12] present an event based mixed integer linear programming method for resource-constrained project scheduling problem.

Mixed integer programming methods are often used to find cyclic schedules in the cell [9,21] which repeat a fixed sequence of robot moves indefinitely. Even though these methods give mathematically optimal results, they are best suitable for high volume production cells. They require that the robot cell is partially filled and they cannot handle

changes in product type, therefore are not very suitable for low production series.

As discussed earlier in the Introduction, the problem presented in this paper is also a symbolic AI planning problem. Classical symbolic planning algorithms such as STRIPS [8] and Graphplan [3] exist in literature as totally ordered planners. LAMA planner [19], uses heuristic guided best-first search to find a solution with minimum action cost. It pre-processes the problem and extracts landmarks, which are propositions that must be true in every plan. The algorithm later uses these landmarks as subgoals.

LAMA employs a forward state space search by imposing a total-order on the actions. This avoids the need for explicit search on conflicts in the plan. A totally ordered plan, means that a new action can start only after the previous one finishes. However, solving a robot cell scheduling problem requires planning many actions that can be executed in parallel. Unlike state space planners, partial order planners [16,20] can order actions in parallel with each other [15], if the required resources for the actions are mutually exclusive. Partial order planners often use backward chaining, starting from the desired goal state and adding supporting actions to achieve that goal. Backward chaining reduces the search space as it explores only the possibilities that would lead to the goal, with the cost of constantly searching for conflicts between actions in the plan. In order to reduce the search for conflicts, Coles *et al.* [4] present partial order planning with forward chaining.

In comparison to the papers presented above, POPStar deals specifically with the robot cell scheduling problem. The proposed algorithm can handle dual gripper robots, parallel machines and it can schedule for multiple object types. We use a best-first search strategy as in [7] and the LAMA planner [18]. LAMA, can minimize the total cost of actions, but not the makespan due to its total ordered characteristics. However, POPStar takes advantage of POP's parallel planning capabilities to minimize the makespan of the plan. Similar to LAMA [18] we rely on landmarks to narrow the search space, but the landmarks instead are partially instantiated actions extracted from user's commands.

3 POPStar

In this section a brief overview of the POP algorithm is given. It is followed by our modifications to solve the programming of a robot cell.

3.1 Partial Order Planner (POP)

A partial order plan can be represented by a set of actions $\mathcal{A}$, a set of constraints $\mathcal{O}$ and a set of causal links $\mathcal{CL}$, together forming a tuple $\langle \mathcal{A}, \mathcal{O}, \mathcal{CL} \rangle$. Each action a in $\mathcal{A}$ is an instantiation of an action A scheme which is defined in the planing domain. A plan may contain multiple instances of an action [20]. Every action is represented

Figure 1: Landmarks graph together with ordering constraints.

using the STRIPS [8] representation, where each actions has a list of *preconditions* and a list of *adds* and *deletes*. The actions in the plan are ordered by a list of partial constraints. A partial constraint $a_i < a_j$ forces the action a_i to occur before action a_j [17].

A causal link, $a_i \xrightarrow{q} a_j$, represents a commitment by the planner that precondition q of action a_j is fulfilled by an effect of action a_i . An *open condition*, $\xrightarrow{q} a_j$, is a precondition q of action a_j that has not yet been linked to an effect of another action [20].

The planner starts with a set of initial conditions $\mathcal{I}$, and a set of goals $\mathcal{G}$. For uniformity $\mathcal{I}$ and $\mathcal{G}$ can be treated as actions $a_{\mathcal{I}}$ and $a_{\mathcal{G}}$ ordered with a partial ordering constraint such that $a_{\mathcal{I}} < a_{\mathcal{G}}$. All new actions that are added to the plan must come after $a_{\mathcal{I}}$ and before $a_{\mathcal{G}}$. When all the open conditions in the plan are resolved, the planning problem is solved [17].

At each iteration in the planning process, an open condition q of action a_j is chosen from the plan. Then the planner searches for an action a_i , that achieves q . Action a_i is either already added to the plan in previous iterations or it is a new action that must be added to the plan. A new constraint $a_i < a_j$ is added to $\mathcal{O}$ and a new causal link is added to $\mathcal{CL}$ which states a_i achieves q for action a_j . If this causal link is threatened by other actions, it is protected by adding extra constraints.

3.2 POPStar

The makespan is defined as the time difference between start and finish of a sequence of activities, in our case performed by an industrial robot. The makespan is a function of the order of the actions that the planner is committing to. The POPStar algorithm relies on the best-first search strategy to choose the consequent action.

POPStar uses STRIPS representation for actions with the addition of temporal information to keep track of starting and processing time of the actions. It is as well possible to use a lookup table to determine the processing time based on variables. This is useful for assigning custom processing times, e.g. different jog times between machines, different processing times for different types of objects.

As in all search problems, it is of vital importance to reduce the search space. POPStar algorithm reduces the search space by allowing the user provide landmarks. Hoffman *et al.* [11] defines landmarks as facts that must

Figure 2: Two consecutive stages of the plan, including a state change.

be true in a given planning problem.

In the cell scheduling case, user’s commands make suitable landmarks, e.g. “pickup an object from the input palette” can be a landmark represented by a partially instantiated pickup action. In this specific case, the *pickup* action has a location variable that is bound to a location, the input palette. It also has several other variables that are not yet bound, such as the robot and the gripper. Landmarks can be used for transforming the user’s commands into a flow shop problem as well. Assume that the following set of landmarks $\{L_1(O_1), L_2(O_1), \dots, L_N(O_1)\}$ are extracted from user instructions for object O_1 . For landmarks of an object O_i the following constraints $L_{j-1}(O_i) < L_j(O_i)$ and $L_j(O_{i-1}) < L_j(O_i)$ should also be added to the constraint list (Fig. 1).

POPStar maintains a state which is a set of facts. It also includes pointers to operators that achieve those particular facts. This representation improves the algorithm’s efficiency in satisfying the preconditions of the operators. For an open condition, $\xrightarrow{q} a_j$, first, POP iterates through all actions in $\mathcal{A}$ to find an action a_i that achieves $\xrightarrow{q} a_j$, if the planner cannot find a suitable action, then a new instance of an action that satisfies $\xrightarrow{q} a_j$ is added to the plan. POPStar would only try to resolve the open condition by unifying it against possible facts in the state, if no fact in state achieves the open condition, then a new action is added to the plan. If all preconditions of an action are met, then the state is updated to encapsulate the effects of the action. Whenever the state is modified the planner will try to expand by choosing between the next possible unachieved actions in the plan.

Two consecutive stages of a plan are shown in Figure 2. The solid line represents the current state, and actions a_1 and L_1 achieve this state (Fig. 2(a)). Among two possible choices L_2 and L_3 to commit to next, the heuristic function has guided the planner to commit to L_3 , and action a_2 has been added to support it. In the next phase (Fig. 2(b)), all preconditions of action a_2 have been satisfied and the state has been modified to reflect changes performed by action a_2 . Newly achieved state and the plan are then added to the open list and in the next iteration, the

Algorithm 1 POPStar Algorithm

```

1: Global variables:
2: open_list
3: closed_list

4: procedure EXPAND( $G_i, a_j$ )
5:    $l \leftarrow G_i$ 
6:   repeat
7:      $G \leftarrow \text{POP}(l)$ 
8:      $q \leftarrow \text{NEXT\_UNSATISFIED\_CONDITION}(a_j)$ 
9:      $\text{solutions} \leftarrow \text{USE\_EXISTING\_ACTION}(G, a_j, q)$ 
10:    for all  $s'$  in solutions do
11:      if  $s'$  changes  $S'_i$  then
12:        add  $s'$  to  $\text{open\_list}$  with value  $f(s)$ 
13:      else
14:        add  $s'$  to  $l$ 
15:    if  $\text{solutions} = \emptyset$  then
16:       $l \leftarrow l \cup \text{ADD\_ACTION}(G, a_j, q)$ 
17:    until  $l$  is empty

18: function SOLVE( $G_0$ )
19:    $\text{closed\_list} \leftarrow \emptyset$ 
20:    $\text{open\_list} \leftarrow G_0$ 
21:   repeat
22:      $G_i \leftarrow \text{POP}(\text{open\_list})$   $\triangleright$  Get lowest-valued successor
23:      $\text{closed\_list} \leftarrow \text{closed\_list} \cup \{G_i\}$ 
24:     if  $\mathcal{W}_i = \emptyset$  then
25:       return  $G_i$   $\triangleright$  Return solution graph
26:     for all  $a_j \in \mathcal{W}_i$  do  $\triangleright$  Expand every possible next action
27:       EXPAND( $G_i, a_j$ )
28:   until  $\text{open\_list}$  is empty
29:   return fail

```

planner will have to choose whether to continue committing to partially achieved action L_3 or start committing to L_2 . Both possibilities will be explored and their successors will be added to the open list.

The algorithm employs open and closed lists to keep track of most promising plans and already tried plans. At any given iteration i the tuple $\langle \mathcal{A}_i, \mathcal{O}_i, \mathcal{CL}_i, \mathcal{S}_i, \mathcal{W}_i \rangle$ is a graph G_i where $\mathcal{S}_i$ is the state of the plan and $\mathcal{W}_i$ is a list of possible next actions to commit to. As an example, for the graph in Figure 2(a), $\mathcal{W}_i$ will include the values L_2 and a_2 .

As shown in Algorithm 1, main loop (lines 18-29) initially starts with P_0 . At every iteration the best scoring graph will be selected from the open list and the planner will try to expand every action in $\mathcal{W}_i$ using the EXPAND procedure. EXPAND procedure will try to find all solutions that will achieve the open condition q of action a_j (lines 4-17). The planner first checks if q can be achieved using an existing action. Any graph where the state has changed is added back to the open list. Later the best first search determines which one to continue with. If an already existing action cannot support the precondition, a new action will be added to support it.

The best first search algorithm is guided by a heuristic function, $f(x)$, which calculates a value for each proposed

Figure 3: A sample plan demonstrating the idle machine times t_m and idle object times t_o , as well as the last action done by the robot t_r .

partial plan. This value represents the quality of that plan. The function is defined as follows: $f(x) = g(x) + h(x)$ where $g(x)$ measures the quality of the proposed partial plan and $h(x)$ represents the distance from the final goal. Plans with a lower $f(x)$ value will have a better makespan.

There are two main factors that contribute to a plan that has a short makespan, which is obviously preferred. These are efficient usage of the robot and the machinery in the cell. Since the robot is responsible for transferring parts between the machines, efficient usage of the robot will lead to minimal makespan. If the robot can accomplish more of the landmarks in shorter time, it leads to an optimal plan.

The second contributor is the efficient usage of the machinery. The robot should minimize the idle time of the machines while transferring the parts. As an example, if the robot has two grippers, when it is unloading a processed object from a machine, it could use the second gripper to load a new object into the same machine before moving to any other stage. Plans that keep the machines idle should be penalized. However, if a machine needs to be idle due to the difference of processing times between two consecutive stages in the manufacturing process, e.g. if a part is waiting for the next machine to finish, then such idle times of the machines should not be used as penalty.

Idle time is calculated as follows; for every process operator which processes part i on machine m , the idle time is the minimum of t_m and t_p (Fig. 3). The first time component, t_m is the time between finishing of processing part $i-1$ on machine m and beginning processing part i on machine m . The second component, t_o is the time between finishing of part i on machine $m-1$ and beginning of processing of part i on machine m . There is no idle time penalty for a new part that is entered into the cell. This is to ensure that bringing new parts can be delayed until needed.

The $h(x)$ part of the heuristic function is calculated as the summation of all the remaining actions processing times based on the landmarks graph for every action that the planner had not achieved yet. For every action that the planner achieves, the processing time of that action is reduced from $h(x)$.

Through linking different landmark graphs (Fig. 1) with ordering constraints, the planner can be directed to

Figure 4: Different landmark topologies.

do many things. The planner can be set to produce multiple types of objects simultaneously (Fig. 4a); while producing one type it can have a smooth transition to another type (Fig. 4b); or a new product type can be introduced into the cell while another type is being produced (Fig. 4c).

4 Results

To demonstrate that the proposed planner is capable of solving scenarios that lack a trivial solution, yet are most likely to happen in a robot cell, a number of cases are presented.

For each scenario, the initial state of the planner is defined using logic predicates. Robot's initial position is declared by a predicate e.g. *at(robot, I)*, stating that the robot is at the input palette. The number of grippers and their status are defined using predicates *holding(robot, G1, nothing)* and *holding(robot, G2, nothing)* for a dual gripper robot. More grippers can be added in the same fashion. The status of the machines in the cell are defined through predicates for every machine e.g. *has(m1, s1, nothing)*. This states that machine m_1 is free for processing an object in stage s_1 . If there are two parallel machines for stage s_i , these machines will be named m_{i-1} and m_{i-2} . Finally the initial locations of the parts are defined e.g. *has(I, Obj1)*. As the plan progresses, some facts from the initial state will be removed and new ones will be added depending on the add and delete effects of operators (Algorithm 2).

In the planning domain four possible actions are defined: *jog*, *pickup*, *load*, *process*. Using these actions it is possible to generate plans to operate the robot cell. The preconditions and effects of actions (Algorithm 2) guide the planner to move the robot to desired machines, load and unload the objects to and from machines, and process the objects in the machines. The load and pickup operators have two overloads: load/unload to a machine, and placing the object at the input/output palette. As an example, the *load* action takes the instance of the robot, the gripper, name of the machine, the stage of the machine and name of the object instance as parameters (Algorithm 2). In the operator definitions the keywords starting with '?' are named variables which are resolved by the planner as planing progresses.

As mentioned in the Introduction part, creating robot

Algorithm 2 Operators

```
jog(?robot, ?x, ?y)
  pre: at(?robot, ?x)
  add: at(?robot, ?y)
  del: at(?robot, ?x)

pickup(?robot, ?gripper, ?mac, ?loc, ?obj)
  pre: object(?obj) & has(?mac, ?loc, ?obj) &
        processed(?obj, ?loc, ?mac) &
        holding(?robot, ?gripper, nothing) &
        at(?robot, ?mac)
  add: holding(?robot, ?gripper, ?obj) &
        has(?mac, ?loc, nothing) &
        holding(?robot, ?obj)
  del: holding(?robot, ?gripper, nothing) &
        has(?mac, ?loc, ?obj) &
        processed(?obj, ?loc, ?mac)

load(?robot, ?gripper, ?mac, ?loc, ?obj)
  pre: object(?obj) &
        holding(?robot, ?gripper, ?obj) &
        has(?mac, ?loc, nothing) &
        at(?robot, ?mac)
  add: holding(?robot, ?gripper, nothing) &
        has(?mac, ?loc, ?obj)
  del: holding(?robot, ?gripper, ?obj) &
        holding(?robot, ?obj) &
        has(?mac, ?loc, nothing)

process(?mac, ?loc, ?obj)
  pre: has(?mac, ?loc, ?obj)
  add: processed(?obj, ?loc, ?mac)
```

programs based on user's commands means that the planner starts from an incomplete set of ordered instructions. Below is a sample of actions for a very simple machine tending example.

```
pickup(?, ?, I, Obj1)
load(?, ?, ?, s1, Obj1)
load(?, ?, O, Obj1)
```

Obj1 is picked up from Input palette referred to as *I*, loaded to a machine to be processed at stage s_1 . Finally the object is dropped at *O*, which is the Output palette. These commands constitute the basis for the landmark concept. To create the full landmarks graph, these commands are copied for N many objects as explained in Section 3. The '?' signs denote unnamed variables. As an example for the first *pickup* action, the system knows that *Obj1* must be picked up from the Input palette, however, it does not know which robot and which gripper should be used to perform this action.

Planner results are presented using Gantt charts (Fig. 5). In these charts each object is represented by a unique color. Rows represent different resources in the cell including the machines $m_1 \dots m_M$. Occupancies of the grippers of the robot are represented using $o.r_1-G_1$ and $o.r_1-G_2$.

The test cases are based on a robot cell where there is a dual gripper robot and a total of 10 machines in 8 stages where machine m_i processes stage s_i . The processing times for these stages are chosen randomly and are 40, 160, 56, 44, 120, 50, 40 and 48 seconds respectively. Stages 2 and 5 have 2 parallel machines and denoted as m_{2-1}, m_{2-2} for stage 2 and similarly for stage 5. A part

Figure 5: Gantt charts for the cases for (a) a robot cell that produces single type object , (b) a robot cell which produces 2 types of parts, and (c) a cell where production changes between different part types. Rows represent machines m_1 to m_M and the actions of the robot and grippers r_1G_1 and r_1G_2 . Bars represent, particular object being processed in a machine.

can be processed in either one of these machines. For simplicity, jog times of the robot between the machines have a constant value of 2 seconds. Loading and unloading also take 2 seconds.

The computations are carried on a core i7 CPU running at 1.7 GHz with allowed memory up to 1.7 GB. The planner runs on a single thread.

4.1 Single Object Case

As the first case, a simple robot cell is presented where one type of object is processed through stages s_1 to s_8 (Fig. 5a). This simple case demonstrates the behavior of the whole system when solely one object type is processed. Thus, this case can be seen as the base line. The total processing time for 16 objects was 48 seconds. As can be seen in Figure 5a the planner has utilized both parallel tracks of stage 2, referred to as $m_{2.1}$ and $m_{2.2}$, as much as possible. Stage 2 is the bottle neck as it has the longest processing time, therefore its maximum utilization is preferred.

4.2 Multiple Part Types

In this case, which is based on the configuration presented in Figure 4a, two different types of objects are to be processed in the robot cell simultaneously. Object type one, represent by solid colors, is to be processed through stages s_1, s_2, s_3, s_4, s_5 and object type two, represented by hatched colors, is to be processed through stages s_4, s_2, s_3, s_5 . Notice that stages s_2, s_3, s_4 are shared between the two object types. The resulting plan (Fig. 5b) can be considered optimal since the machines $m_{2.1}$ and

$m_{2.2}$ which are parallel machines and are the shared machines for both object types which have the longest processing time, have been utilized optimally. The processing time for a total of 24 objects took 365 seconds.

4.3 Changing Part Types

This case is based on the configuration presented in Figure 4c and considers an urgent low volume production where another object type is introduced into an already fully operating cell. Thus, while the cell is producing one type of object, it is asked to process a second type of object. As mentioned in Subsection 3.2, this is achieved by adding a new group of landmarks into the plan. For this cell, object type one and two go through stages s_1, s_2, s_3, s_4, s_5 and s_4, s_2, s_3, s_5 respectively. The results are presented in Figure 5c. The planner can incorporate the second object type into the plan, while keeping track of the objects which are already in the pipeline. For a while the cell produces both object types simultaneously. When processing of all objects of type two is done, the cell switches back to processing solely type one again. The processing time for a total of 32 objects took 541 seconds.

5 Conclusion

In this paper, robot scheduling problem in which different types of objects are processed in the same robot cell is solved using the POPStar algorithm. The problem includes cases where two types of objects are processed in the robot cell simultaneously. This means that these two

types of objects are processed in different stages while having some stages in common. POPStar planner generated optimal plans for these cases, since the bottleneck stages are used efficiently as new object types are introduced into the cell.

The POPStar algorithm uses principles from partial order planning and uses heuristics to guide the planner to create complete plans while minimizing the makespan. The advantage of POPStar against other solutions is that it uses first-order logic to create the constraints in the search space through the definitions of actions, therefore it is simpler to define and extend the problem to different types of setups.

The work presented in this paper can be further improved in many ways and more quantitative tests need to be performed. A load balancer can be added to the planner so that different production rates can be assigned to different object types. This would increase the user's possibility to control the robot cell even further. It is also possible to execute the planner in an interactive manner on a robot cell while the robot cell is operating. This will give the user the ability to handle changes which might be required in a robot cell, interactively. This offers the flexibility that is of great importance to an SME.

References

- [1] B. Akan, A. Ameri, B. Çürüklü, and L. Asplund. Intuitive Industrial Robot Programming Through Incremental Multimodal Language and Augmented Reality. In *International Conference on Robotics and Automation*, 2011.
- [2] A. Ameri, B. Akan, and B. Çürüklü. A general framework for incremental processing of multi-modal inputs. In *Proceedings of the 13th International Conference on Multimodal Interaction*, ICMI'11, 2011.
- [3] A. Blum. Fast planning through planning graph analysis. *Artificial Intelligence*, 90(1-2):281–300, Feb. 1997.
- [4] A. Coles, A. Coles, M. Fox, and D. Long. Forward-Chaining Partial-Order Planning. In *Twentieth International Conference on Automated Planning and Scheduling*, 2010.
- [5] M. Dawande, H. N. Geismar, S. P. Sethi, and C. Sriskandarajah. Sequencing and Scheduling in Robotic Cells: Recent Developments. *Journal of Scheduling*, 8(5):387–426, Oct. 2005.
- [6] K.-H. Ebel. The impact of industrial robots on the world of work. *Robotics*, 3(1):65–72, Mar. 1987.
- [7] J. P.-o. Fan and G. K. Winley. A heuristic search algorithm for flow-shop scheduling. *Informatica*, 32:453–464, 2008.
- [8] R. E. Fikes and N. J. Nilsson. STRIPS : A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence*, 2, 1971.
- [9] H. N. Geismar, M. W. Dawande, and S. P. Sethi. Dominance of Cyclic Solutions and Challenges in the Scheduling of Robotic Cells. *SIAM Review*, 47(4):709–721, Jan. 2005.
- [10] H. N. Geismar, M. Pinedo, and C. Sriskandarajah. Robotic cells with parallel machines and multiple dual gripper robots: a comparative overview. *IIE Transactions*, 40(12):1211–1227, Oct. 2008.
- [11] J. Hoffmann and B. Nebel. The FF Planning System: Fast Plan Generation Through Heuristic Search. *Journal of Artificial Intelligence Research*, 14(1):253–302, 2001.
- [12] O. Kon. Event-based MILP models for resource-constrained project scheduling problems. *Computers and Operations Research*, 1:3–13, 2011.
- [13] E. Levner, V. Kats, D. Alcaide López de Pablo, and T. Cheng. Complexity of cyclic scheduling problems: A state-of-the-art survey. *Computers & Industrial Engineering*, 59(2):352–361, Sept. 2010.
- [14] G. Michalos, S. Makris, N. Papakostas, D. Mourtzis, and G. Chrysosouris. Automotive assembly technologies review: challenges and outlook for a flexible and adaptive approach. *CIRP Journal of Manufacturing Science and Technology*, 2(2):81–91, Jan. 2010.
- [15] X. Nguyen and S. Kambhampati. Reviving partial order planning. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence - Volume 1, IJCAI'01*, pages 459–464, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.
- [16] J. Penberthy and D. Weld. UCPOP: A Sound, Complete, Partial Order Planner for ADL. *KR*, 1992.
- [17] D. Poole and A. K. Mackworth. *Artificial Intelligence - Foundations of Computational Agents*. Cambridge University Press, 2010.
- [18] S. Richter and M. Westphal. The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research*, 39:127–177, 2010.
- [19] S. Richter, M. Westphal, and M. Helmert. LAMA 2008 and 2011. *The 2011 International ...*, 2011.
- [20] H. Younes and R. Simmons. VHPOP: Versatile heuristic partial order planner. *J. Artif. Intell. Res.(JAIR)*, 20:405–430, 2003.
- [21] Z. Zhou, A. Che, and P. Yan. A mixed integer programming approach for multi-cyclic robotic flowshop scheduling with time window constraints. *Applied Mathematical Modelling*, 36(8):3621–3629, Aug. 2012.