Online Planning to Control a Packaging Infeed System

Minh Do, Lawrence Lee, Rong Zhou, Lara Crawford, and **Serdar Uckun**
Palo Alto Research Center
3333 Coyote Hill Road, Palo Alto, CA 94304
minh.do, *lawrence.lee*, *rzhou*, *lcrawford*, *uckun* at *parc.com*

Abstract

In this paper, we investigate a novel application of online planning and scheduling: controlling an automated infeeder for a packaging line of food and consumer packaged goods. In this system, products arrive continuously at high-speed from the end of the production line and need to be arranged into a specific configuration for downstream primary and secondary packaging machines. In collaboration with a domain expert from the packaging industry, we developed an innovative design for a reconfigurable parallel infeed system using a matrix of interchangeable smart belts. We also adapted our online model-based Plantrol planner to this domain. Our planner can control various configurations of the new infeed system through simulation both in nominal planning and when runtime failures occur. We are also building a small physical prototype to validate the new design and our software framework.

Introduction

The Tightly Integrated Parallel Printer (TIPP) project at PARC (Ruml *et al.* 2005; Do *et al.* 2008) shows that general purpose online planning algorithms can successfully control a complex production manufacturing system. To utilize our Plantrol software framework started by the TIPP project, the Embedded Reasoning Area (ERA) at PARC has been investigating new applications that share similar characteristics. In this paper, we describe our recent effort in one such application: high-speed packaging machines. Specifically, we address the problem of controlling the infeed system of the flow-wrapper machines.

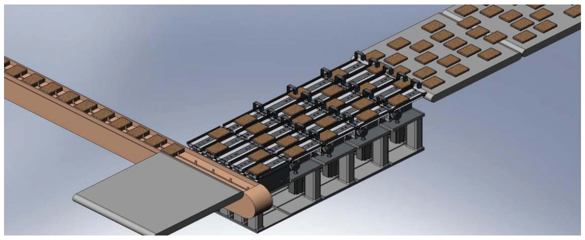
In the food industry, an automated packaging line normally consists of several main components:

- *The conveyor from the production line:* The items that need to be packaged come out from the end of a production line or a storage system such as a freezer. When transported on the conveyor, they can often shift out of alignment, both transversely in rows and inline in lanes.
- *The infeed system:* The items are then transferred into an infeed system that can collate them into a well-aligned pattern that is synchronized with a downstream packaging machine such as a flow wrapper.
- *The packaging line:* The flow wrapper wraps the items in a specified configuration to create the packaged product that can be transferred to cartoning, case-packing, and other secondary and tertiary packaging machines to complete the packaging process.

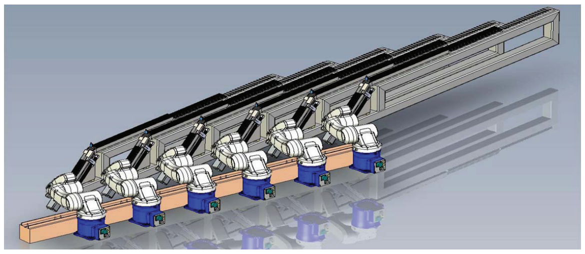
Depending on how the output of the production machine and the input of the flow wrapper are specified, there can be different designs for the infeed system. There are several problems with the typical existing cross-feeder: (1) they tend to jam when products are not aligned well; (2) they do not have active control over the products so that an extra cross-feeder and packaging line are needed to act as spare or backup capacity for temporary increases in volume from the production line; (3) they are not easily reconfigurable to handle different wrapping configurations; (4) they have a large footprint.

To overcome these drawbacks, we have developed a new infeed system design that assimilates TIPP's modular structure. More specifically, a new infeed system is made up of a matrix of interchangeable smart belt modules that can be controlled individually. Figure 1(a) shows one example of the design. In this figure, products come out of the production line or storage system from the right in an unordered manner and enter the infeed system, which in this case is made up of a 5×5 matrix of smart belt modules. By controlling the speed and acceleration of each smart belt when a given product is on it, the matrix is able to collate and synchronize multiple parallel output lanes to achieve a desired configuration. Advantages of the new design are: (1) higher flexibility to create many different modular versions of the infeed system simply by deciding the dimensions of the matrix and the specification of each smart belt module; (2) higher operational efficiency: any lane can be taken offline for cleaning or service while other lanes continue operating (instead of requiring a complete shutdown of the packaging line); (3) higher reconfigurability: products can be collated for any possible wrapping configuration without re-programming; (4) smaller footprint due to an inline design that not only reduces the space needed for a right-angle transfer but also eliminates the need to keep a spare infeed and packaging line.

One of the keys to the success of this new design is the ability to control the matrix composed of individual modules. This is a challenging problem given that the incoming rate of products can be *very high*, up to several hundred per minute per lane and the products are not well-aligned in rows. For this task, we have adapted our fast online Plantrol planner, originally developed for the TIPP project. Our results in simulation confirmed that our online



(a) Modular smart-belt infeed system



(b) Modular infeed system with robotic-hand as end-effector

Figure 1: Exemplary modular smart-belt infeed system designs with either stacking or pick-and-drop end finish.

planner can indeed successfully control various modular infeed system designs.

Project History: In 2007, upon discussion with experts in the field, we identified advantages for modular packaging machine designs similar in spirit to the TIPP printer. In late 2007 we invited a domain expert with a long history of building packaging machines to be a paid Entrepreneur in Residence (EIR) at PARC to co-develop machines with this new modular design. While our EIR had experience in hardware and customized control software for individual machines, he had no experience in model-based software that works for multiple reconfigurable configurations of this complex nature. After he learned about our software's capabilities, our EIR identified that infeed collation for primary packaging lines would be the highest value problem we could solve in consumer packaging. While he was able to confirm hardware feasibility for the design concept (machines using a single line of smart belts had been previously built), our EIR did not know how our software would operate in a multiple line array configuration. As a result, we created software simulations for multiple configurations of the novel design based on the hardware specifications/constraints given by our EIR in order to convince ourselves of the feasibility of the complete system. In 2008 we attended *PackExpo* and *Interpack*, which are the main U.S. and international consumer packaging tradeshows to do further market and competitive analysis. In this time period we also presented the whole concept, including software simulations, to two large food packaging companies with positive feedback.[1] Starting from 2010, we have been building a simple hardware prototype to further test the concept. However, this process has been slow due to personnel turnover and several other difficulties. We will elaborate on our current results in the later part of this paper.

[1]Due to confidential information, we cannot list the names of the companies.

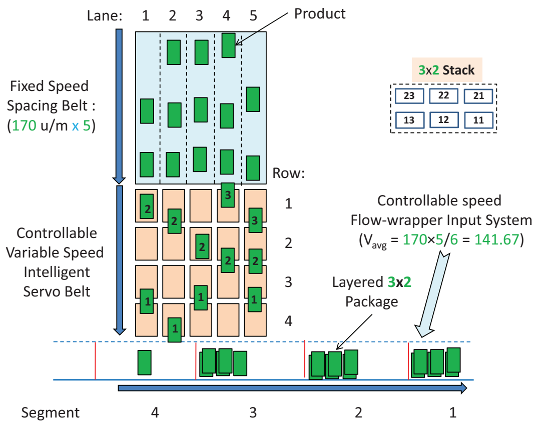

Lane: 1 2 3 4 5
Product
Fixed Speed Spacing Belt : (170 u/m x 5)
3x2 Stack
23 22 21
13 12 11
Row:
1
2
3
4
Controllable speed Flow-wrapper Input System (V_{avg} = 170×5/6 = 141.67)
Controllable Variable Speed Intelligent Servo Belt
Layered 3x2 Package
Segment 4 3 2 1

Figure 2: Detailed description of a modular infeed system

Intelligent Infeed System: Design and Control

Figure 1(a) shows one example of the infeed system using our modular design. A more complex system may have several infeeds of the same design sharing the input from the production machine. Other designs may have different end-effectors that can collate different types and quantities of products for different packaging configurations (Figure 1(b) shows one such example). We will use the design outlined in Figure 1(a) as the running example.

Figure 2 shows a more detailed view of our leading example system. Products enter the infeed system from a fixed-speed *spacing belt*. While the average incoming rate for each lane is known (170 units/minute/lane in this example), the exact timing when each product starts to enter the smart belt system is unknown. There are photosensors placed at the entry point of each smart-belt module to recognize the time instant at which a product enters each module. For the first row of modules (just next to the spacing belt), the sensor feedback provides the timing information when new products enter the system.

The flow-wrapper input system, which is at the bottom of Figure 2, consists of continuously arriving empty *segments* on a belt and the speed of this input system is also controllable. The flow wrapper is preconfigured to wrap packages as specified by the operator of the infeed system. In our example, each package is a stack of 3×2 products (three stacks of two units each). Each segment needs to either be filled up to the specification (i.e., in a 3×2 stack) or left empty. Given that each lane only aligns with a given slot in each segment (there are 6 slots in a 3×2 package configuration) at a brief moment in time, the planner needs to decide when to speed up or slow down different product units by adjusting the speed of each of the four belt modules that each product will pass through so that it will arrive at the wrapper-input line at the exact moment when that intended segment/slot is aligned with that lane. One feature that is not clear from Figure 1(a) and Figure 2 is the ability to skip a given product by making it "bypass" the wrapper input. This is done by mechanically extending the last belt module over the top of the wrapper input (this is a common hardware design in the food packaging industry).

The main objective function is to minimize the total number of empty segments with the following constraints:

- No partially filled segments.
- There are limits on the maximum acceleration or deceleration of each module.
- For more precise control and tracking, multiple products cannot be on the same module at any given time.

In summary, the planning/scheduling problem is to control the movement of known products through the smart belt modules so that collectively they are synchronized to fill up every segment of the wrapper input system, obeying all constraints listed above. Given that the overall path of each product is well-defined and the only choice is whether or not to "bypass" a product, this problem has more of a scheduling flavor than planning. The main challenge is that the products arrive randomly at high speed and high volume (up to 1000+ products/minute) and all need to be controlled individually within the constraints listed above.

The Plantrol Framework

Starting from TIPP (Ruml *et al.* 2005; Do *et al.* 2008), the Embedded Reasoning Area at PARC has been building the Plantrol framework that tightly integrate high-level planning and low-level control. We believe it is applicable in a wide range of on-line continual decision-making settings; the packaging machine control problem described in this paper is one of several applications that we are investigating. While the framework is designed to be general and applicable to multiple applications, certain adaptation to a specific domain is needed at both the planning and control levels. However, the adaptation effort is meant to be minimal compared to building customized software for each domain instance. For the rest of this section, we will discuss how we adapted Plantrol's planning component to the intelligent infeed system. In a later section, we will outline the current work on the prototype hardware and low-level control framework.

Adaptation to Packaging Domain: While both this and the TIPP application are of online-continual production nature, there are a couple of key differences between them: (1) the packaging machines contain fewer parts and the product paths are simpler with less interactions; (2) the productivity of packaging machines are higher; and most importantly (3) while we can control when to feed a paper in the printer domain, we can not control the incoming food product. If we stop the printer or take too much time to find the plan in TIPP, we only suffer lower productivity. However, the food products will crash/jam if the infeeder is stopped or the controller is slow to respond. The last two differences enforce the planner to produce valid plans with less amount of time and at a much more consistent planning time.

Figure 3 shows the pseudo-code of the overall Plantrol planning framework, as outlined above, adapted to this packaging-infeeder control domain. Specifically, lines 1-9 outline the main steps within the Plan Manager to receive online messages and call appropriate functions to either conduct nominal planning or replanning to fill in all incoming segments. The rest of this section will concentrate on the two key functions: (1) *PlanSegment* that finds a plan/schedule to fill up an individual segment with arrived but unplanned products; and (2) *Replanning* that handles real-time failures/repairs.

```
On-line Planner for Modular Packaging Infeed System
01. repeat
02.    Analyze new online message
03.        if new product arrival message
04.        then update the list of unplanned products
05.        if failure/repair message
06.        then call Replanning function
07.    Call the PlanSegment function
08.    Send the found plan to the machine controller
09. until stopped

PlanSegment: heuristic search to fill in segment $S$
10. if not enough unplanned products then return no-plan
11. Order the set of unplanned products $P$
12. repeat
13.    pick the next unplanned products $p_i$ in (ordered) $P$
14.        branch over unfilled slots $l$ in $S$ eligible for $p_i$
15.            assign $p_i$ to $l$ and post temporal constraints
16.            backtrack if temporal constraint violation
17. until $S$ is filled or search-space exhausted
18. if $S$ is filled then return the complete assignments
19. else return no-plan

Replanning: handle failure/repair messages
20. Adjust the wrapper-input speed
21. Replace the planned products lost due to failure
22.    Call PlanSegment for affected segments
23. Rebalance for the incoming segments
```

Figure 3: Plantrol planning adapted to packaging domain

Nominal Planning

In nominal planning, the planner is (continually) given a goal of filling an arriving segment. Figure 4 shows an exemplary stage in this continual planning process. At this stage, we have already finished planning/scheduling for segments #1-4 and are planning to fill up the next arriving segment #5. At any given time, the planner knows about all products that are currently on the smart-belt matrix (i.e., all numbered products with backgrounds in black or white). Among those, all white-background products are already scheduled to be put in slots in segments #3 and #4 and so are not available to be candidates for segment #5. The planning problem is to find a subset of 6 candidate products such that: (1) they can fill out all slots in the wrapper-input segment #5 and (2) all the constraints listed in the previous section are satisfied.

The main steps of the planning search algorithm are outlined in line 10-19 of Figure 3.

Defining Planning Search Space: Our planner/scheduler uses heuristic search. Therefore, we need to define the root node, the expansion function, and the goal satisfying conditions to terminate the search.
Root Node: The initial planning state (i.e., root search node) contains information on: (1) all candidate products and their locations; (2) all 6 slots 11-23 (referring to the right side of Figure 4) are empty; (3) a Simple Temporal Network (STN) represents all temporal constraints.

Expansion/Heuristic: Our current approach first statically orders all known candidate products P (line 11 in Figure 3) and then branches over feasible slots that P can fill in (line

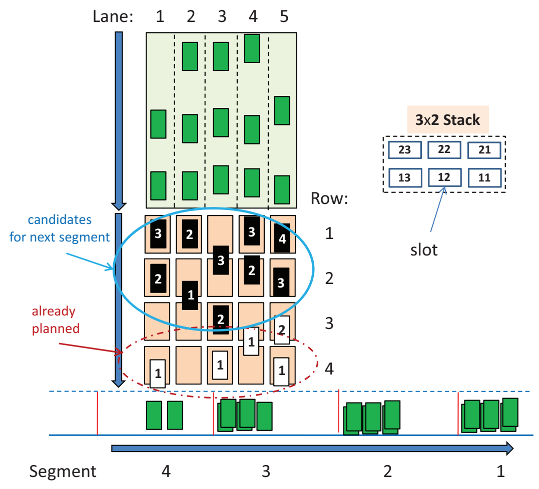


Figure 4: A planning stage example.

13 in Figure 3). Thus, expanding a search node at depth i corresponds to picking the i^{th} candidate product p_i from P and branching over the k slots that p_i can go in plus 2 additional branches for bypassing or saving p_i for later segment. Heuristics for ordering search nodes are:

- *Candidate product ordering:* with regard to all the constraints listed in the previous sections, let $d = [t_s, t_e]$ be the time window bounding the time instant at which a given product p can arrive at the wrapper-input, and t_1, t_2, t_3 be the time instants that slots 11 (and 21), 12 (and 22), 13 (and 23) align with the lane at which p was on. We order the candidate products based on: (1) between products in the same lane: the earlier one is selected first (e.g., on lane 1 in Figure 4, product #2 is ordered before #3); (2) between different lanes: we select the one with the smaller number of candidate slots (i.e., smaller number of cases in which $t_s \leq t_i \leq t_e$ with $1 \leq i \leq 3$).[2]
- *Best-First-Search open-list ordering:* Generated nodes are ordered in the open list with higher priority given to: (1) smaller total number of bypassed products in the path from root node to a given node; (2) break-tie over the higher number of filled slots.

Goal Satisfaction & Final Plan/Schedule: a goal state is reached when all slots in a given segment are filled. At this point, the final destination for the planned products are decided: a particular slot in the arriving segment, bypassing the wrapper-input, or remain unplanned. However, the actual duration each product spends on each belt module is not decided yet and thus some scheduling decisions should be made. We run each planned product as quickly as allowed by the various temporal constraints on the earlier modules and as slowly as possible in the later modules. For example, the product #1 in lane 2 will run as quickly as allowed through row 3 and slow down in row 4. The main purpose is to increase the inter-product gaps and allow more control flexibility for the subsequent unscheduled products.

[2]The second criterion is inspired by CSP's most-constrained-variable-first ordering heuristic.

Temporal Constraints: A Simple Temporal Network (STN) (Dechter *et al.* 1991) manages various time-points and constraints between them. Time points represent the instants at which each product enters or leaves a given smart-belt module. In our ongoing example, for each product p, we create 8 time-points representing its trajectory. There are temporal constraints: (1) between consecutive time-points on a trajectory of each product to enforce maximum/minimum speed and acceleration/deceleration limits; (2) between time-points at which consecutive products enter or leave a given smart-belt module to ensure that only a single product can be on that belt module at any time. For example, if a given product p enters a module s running at speed v at time point t_1, given that we know the minimum speed v_m, maximum speed v_M, and the maximum acceleration/deceleration values a_M, a_m of s, we can calculate the upper and lower-bound values u, l on the duration p spends going through s. We then add the constraint $l \leq t_2 - t_1 \leq u$ with t_2 being the time-point when p leaves s.

Pruning: Given that the system productivity is very high, it's very critical to *consistently* plan really fast for each segment. Therefore, it's useful to have effective pruning techniques to reduce the branching factor:

- For a given product p from the first lane (e.g., #2 on lane 1 in Figure 4), the candidate slots can only be 11, 12, or 13 because to be able to get into 21, 22, or 23, it needs to have some other product already residing on 11, 12, or 13 at the time of p's arrival; which is physically impossible. Similarly, candidate slots for products from the last lane (lane 5) can only be the upper ones: 21, 22, or 23.
- Limit the number of products from a single lane by the number of horizontal slots. In our example, no more than 3 products from a single lane can fill in a single segment.
- Two products from a single lane cannot be stacked on top of each other.
- Two products stacked on top of each other should also arrive in the order that is physically possible: the lower slot should be filled before the top slot.

Exception Handling

An operating smart-belt module can fail or need to be turned off for maintenance. A failed module can also be repaired and put back into operation. When this happens, the entire lane is taken either offline or online and the planner has to transition smoothly to planning for the decreased or increased number of operating lanes, without stopping the whole machine.

We will use as a running example the scenario shown in Figure 5 where the third lane is taken offline due to failure in one of its modules. When failure happens, products on the infeed system from both the *already planned* and *candidates for next segment* groups are affected. This potentially leads to: partially filled segments and/or not having enough candidates to plan for the next empty segment. In our running example, if product #1 in the failed lane 3 was scheduled to fill in segment #4, then if we don't find a replacement, segment #4 will end up as a partially-filled segment. The removal of products #2 and #3 on lane 5 may also lead to not having enough products to fill in the upcoming segment

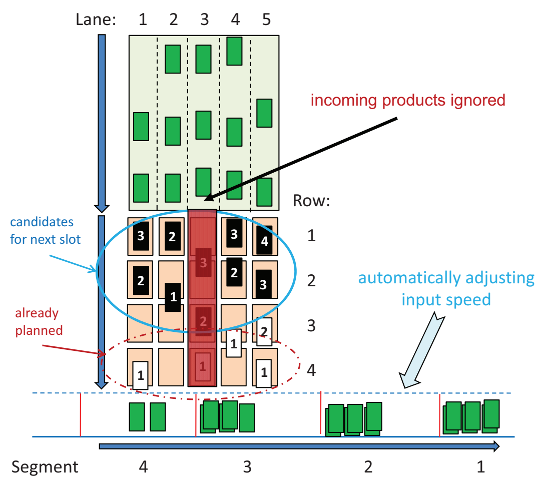


Figure 5: An example of a failure scenario.

#5. Lines 20-23 in Figure 3 shows the main steps for the replanning routines to handle the problems outlined above:
Adjust the wrapper-input speed: Given that the total incoming rate of product decreases as lanes fail, the outgoing rate of product into the wrapper needs to be adjusted accordingly to avoid empty segments. The question is *when* and *how*:

- At wall-clock time t_f when failure occurs, we identify the wall-clock time $t \geq t_f$ at which the last planned product gets into the intended segment.
- Let n be the total number of operational lanes before the failure, $m < n$ be the number of operational lanes after the failure (normally $m = n - 1$), and s be the current speed of the wrapper-input system. We schedule to adjust at t the input speed to $s' = s \times m \div n$.
- Using the speed profile of the wrapper input such that it will go at speed s until t and continue with s' after t, we can calculate the time instants at which each slot in the unfinished segments (#3, #4, and later) align with each of the five lanes. We use that information to replan the remaining candidate products.

Replace the planned products that are lost due to failure: In our example, product #1 on lane 3 was planned but now lost. We find another product from the remaining unplanned products to replace it and then continue with the subsequent segments (exactly like the nominal planning case).
Rebalance for the incoming segments: even if we adjust the speed perfectly, in many failure cases the planner needs to "bypass" some products and leave one empty segment during the transition period and then fill in all subsequent segments. For example, segment #5 is left empty while it bypasses three products from the candidate product set, then fills in all segments starting from #6. Our planner does this automatically.

Our planner operates similarly when a lane is *repaired*, but some steps are done in a reversed manner. For example, the wrapper-input speed is increased instead of reduced.

Empirical Results

We modeled different configurations following the two designs shown in Figure 1. Figure 1(a) contains the main design (which has been used as the running example), which we have tested more extensively (i.e., more variations) while the design shown at the bottom of the figure has been tested with fewer variations. The main difference between them is that the second design has a special finishing component that collates multiple products at the end of each line before dropping them into the wrapper's input line. This design is less flexible, more complex and expensive for hardware but is suitable for some particular food products and also can handle much faster product incoming rates. For the rest of this section, we will refer to the design shown in Figure 1(a) as *Config-1* and the design shown in Figure 1(b) as *Config-2*.
Config-#1: testing scenarios are characterized by:

- *Matrix size*: range from 5×4 to 8×5.
- *Stacking configuration:* tested sizes ranged from 1×2 to 3×3 (our running example is 3×2). The higher the number of slots in the stacking configuration, the more products need to be coordinated.
- *Product incoming rate:* ranged between 60-240 products/lane/minute. Thus, depending on the particular number of lanes, the overall throughput ranged from 300 - 1200 products/minute. The higher the throughput, the less planning/scheduling time is allowed to keep up.
- *Product incoming randomness:* if the average incoming rate is 240 products/min/lane, for example, then if there is no uncertainty, the arrival time of the n^{th} product in a given lane should be $t = n \times 0.25$. Our test generation program randomly injects up to 40% randomness in arrival time. Thus, the n^{th} product will arrive randomly between $(n-0.4) \times 0.25$ and $(n+0.4) \times 0.25$. The higher the randomness, the harder it is to control all products.

Config-#2: In this design, shown at the bottom of Figure 1, we tested with up to 8 lanes and product incoming rates of up to $8 \times 600 = 4800$ products/minute.

The planner can run on either Linux or Windows machines. It is able to handle both nominal planning and replanning for the configurations specified above. For all configurations, we tested in simulation between a few hundreds to few thousands products. The planner can keep up with the high throughput rates in all cases. In nominal planning, the planner never left any segment empty or had to bypass any product. Thus, it allows the system to run optimally. In exception handling mode, the user can randomly fail or repair/restore a lane through a GUI connected in real-time with the planner. In all random test scenarios, the planner leaves at most 1 empty segment during the transition period (i.e., failure recovery duration) and bypasses the minimum number of products.

Figure 6 shows the average running time for the example configuration (i.e., 5×4 stacking into 3×2 segments) with the total product incoming rate of 1200 products/minute for the first 240 products (40 segments). The solving time is quite stable between 0-4 msec/segment, which is more than enough to keep up with the incoming segment rate of 200 segments/minute or 300 msec/segment. Note that the planning time is not important in this domain, as long as the planner can generate plans/schedules fast enough to keep up with the incoming rate. In cases of failures/repairs, the replanning time is very similar to normal planning time.
Visualization: We also developed a visualization tool that works with the Plantrol planner. It can:

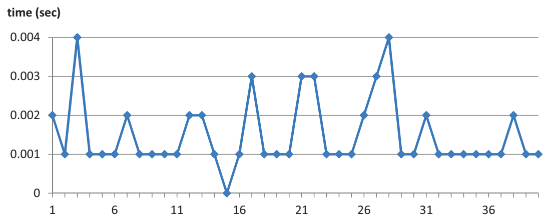
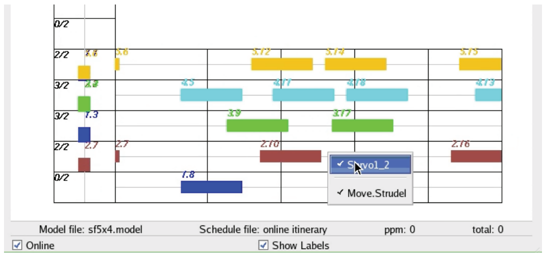


Figure 6: Planning/scheduling time for 5×4 stacking into 3×2 segments with a throughput of $240 \times 5 = 1200$ products/minute at 40% ramdomness. The horizontal axis shows the segment sequencing number and the vertical axis shows the planning time.



Model file: sf5x4.model Schedule file: online itinerary ppm: 0 total: 0

☑ Online ☑ Show Labels

Figure 7: Interactive visualization tool.

- Display in real-time the trajectory of all products according to the plan/schedule found by the Plantrol planner.
- Through the visualizer (Figure 7), we can randomly inject exceptions by "failing" or "repairing" different lanes in real-time. When failure or repair happens, we can also see the adjusted wrapper-input speed, product bypass and transition as described in the previous section.

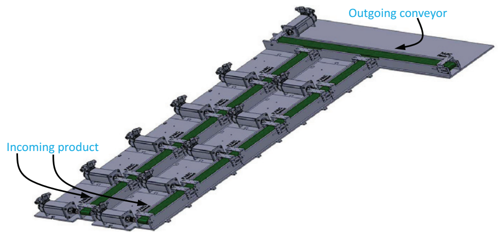
Hardware Prototype: In order to execute the planner-generated product plans in a physical system, the plans must be translated into commands for each servo belt module. Although the planner specifies time points at each servo belt module, it does not specify the velocity profile of the product on the belt, leaving the controller free to define this profile. In addition to meeting the time points imposed by the planner, the controller must take into account speed and acceleration limits of the physical system, which may include limitations imposed by the need to maintain friction or a vacuum connection between the product and the belt.

These issues and others not addressed in simulation, such as network communication and sensing, will be addressed in a modular infeed prototype that is currently under development. This proof-of-concept prototype, shown in Figure 8, will consist of several lanes of belt modules with multiple outgoing conveyors representing inputs to flow wrappers. The servo motors, the most critical components, are from Bosch-Rexroth and follow the industry standard.

Related Work and Discussion

Previous to TIPP, constraint-based scheduling was used for online planning and planning for print shops and offices world-wide (Fromherz *et al.* 1999; 2003). Unlike ours, these approaches use hand-coded domain knowledge to guide the scheduling process. In contrast to much work on continual planning (desJardins *et al.* 1999), the tight constraints and the requirement that all slots in any segment need to be filled require that we produce a complete plan/schedule for each segment before we can send it down to the controller for execution. While the constraint set involved with our domain is quite simple compared to other continuous scheduling applications, such as scheduling for airlift and tanker resources (Smith *et al.* 2004), our domain requires a very fast planning/scheduling time to react to very high throughput and real-time component failures.

Commonalities between different Plantrol applications: The packaging machine discussed in this paper is the second of four applications that we have adapted our fast online planner to solve. The general shared characteristics we observed are: (1) they all require fast planning time (less than 1 second), interleaving goal arrival, planning, and execution; (2) they have "logistic" flavor (which is one of the easier classes of planning problem); (3) makespan optimization; and (4) planning for invididual goals sequentially (instead of concurrently for all known goals) produce good overall quality solutions.



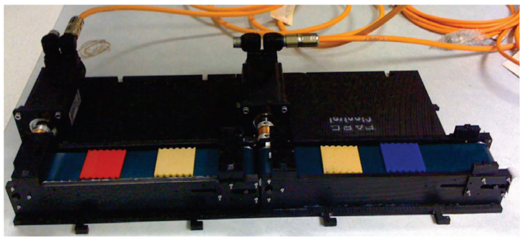


Figure 8: Our packaging prototype.

References

Rina Dechter, Itay Meiri, and Judea Pearl. Temporal constraint networks. *Artificial Intelligence*, 49:61–95, 1991.

Marie E. desJardins, Edmund H. Durfee, Charles L. Ortiz, Jr., and Michael J. Wolverton. A survey of research in distributed, continual planning. *AI Magazine*, 20(4):13–22, 1999.

Minh Do, Wheeler Ruml, and Rong Zhou. On-line planning and scheduling: An application to controlling modular printers. In *Proc. of AAAI08*, 2008.

Markus P.J. Fromherz, Vijay A. Saraswat, and Daniel G. Bobrow. Model-based computing: Developing flexible machine control software. *Artificial Intelligence*, 114(1–2):157–202, October 1999.

M.P.J. Fromherz, D.G. Bobrow, and J. de Kleer. Model-based computing for design and control of reconfigurable systems. *AI Magazine*, 24(4):120–130, 2003.

Wheeler Ruml, Minh B. Do, and Markus Fromherz. On-line planning and scheduling for high-speed manufacturing. In *Proc. of ICAPS-05*, pages 30–39, 2005.

S. Smith, M. Becker, and L. Kramer. Continuous management of airlift and tanker resources: A constraint-based approach. *Mathematical and Computer Modeling – Special Issue on Defense Transportation: Algorithms, Models and Applications for the 21st Century*, 39(6-8):581–598, 2004.