Planning for Modular Printers: Beyond Productivity

Minh B. Do
Embedded Reasoning Area
Palo Alto Research Center
Palo Alto, CA 94304 USA
minhdo at parc.com

Wheeler Ruml
Department of Computer Science
University of New Hampshire
Durham, NH 03824 USA
ruml at cs.unh.edu

Rong Zhou
Embedded Reasoning Area
Palo Alto Research Center
Palo Alto, CA 94304 USA
rzhou at parc.com

Abstract

This paper reports our experience extending an on-line printer controller based on AI planning to handle two significant features of this commercially important domain: execution failures and multi-objective preferences. A printer controller must plan quickly and reliably, otherwise expensive human intervention will be required. Our approach is practical and efficient, and showcases the flexibility inherent in viewing planning as heuristic search. Execution failure is handled by replanning. We link together the individual searches for each in-flight sheet, giving rise to a tree of potentially infinite branching factor. Multiple objectives are handled by linear combination and tie-breaking during best-first search. Multiple pre-computed pattern databases are used to improve the efficiency of handling preferences regarding image quality. Our successful experience controlling multiple prototype printing systems shows that replanning and preference-handling can be made practical without using hand-coded control knowledge.

Introduction

It is a sustaining goal of AI to develop techniques enabling autonomous agents to robustly achieve multiple interacting goals in a dynamic environment. This goal also happens to align perfectly with the needs of many commercial manufacturing plants. In this paper, we focus on one particular manufacturing setting: high-speed digital production printing systems. Unlike traditional continuous-feed offset presses, digital xerographic cut-sheet printers can treat each sheet differently: printing a different image and performing different preparatory and finishing operations. Often, a single integrated machine can transform blank sheets into a complete document, such as a bound book or a folded bill in a sealed envelope. It is sometimes even possible to process different kinds of jobs simultaneously on the same equipment. Designing a high-performance yet cost-effective controller for such machines is made more difficult by the current trend towards increased modularity, in which each customer's system is unique and includes only those components that are most appropriate for their needs. We have been working closely with the Xerox Corporation to explore architectures in which printing systems can be composed of literally hundreds of modules, possibly including multiple specialized printing modules, working together at high speed.

Previous work has shown how AI techniques can be used to control such machines (Ruml *et al.* 2005; Do and Ruml 2006; Do *et al.* 2008). Requests for printed sheets become goals for the system to achieve, the various actuators and mechanisms in the machine become actions and resources to be used in achieving these goals, and sensors provide feedback about action execution and the state of the system. The main objective in previous work has been maintaining high productivity, and thus high return on investment for the equipment owner. While this is the most common and important objective, it is by no means the only thing that owners care about.

In this paper, we address two major challenges. The first is execution failure and exception handling. To reduce the need for operator oversight and expertise and to allow the use of very complex mechanisms, the system must be as autonomic as possible. Because operators can make mistakes and even highly-engineered system modules can fail, the system must cope with execution failure. And because the system must work with legacy modules in order to be commercially feasible, its architecture must tolerate components that are out of its direct control and will give rise to unexpected events. The second challenge is complex objectives. In a modular system with multiple print engines, one might want to optimize the cost of printing by choosing to print black-only pages only on monochrome engines and avoid using more expensive color engines. One might want to optimize image quality by choosing to print pages from the same document only on print engines whose current marking gamuts are similar. The printer controller needs to give operators the ability to trade off these conflicting objectives while maintaining robust operation.

We meet these challenges using (1) fast replanning to handle various types of exceptions in plan execution, (2) multi-objective optimization to handle both productivity and printing cost, and (3) multiple heuristic look-ups to efficiently handle image quality consistency constraints. We conclude with a discussion of our experience using the planner to control three physical prototype printing systems as well as results from simulation studies.

Background

In analogy to other parallel systems such as RAID storage, our approach to modular printing systems is called Rack Mounted Printing (RMP). A modular RMP system can be seen as a network of transports linking multiple printing engines. These transports are known as the media path. Figure 1 shows a four-engine prototype printer located at the Palo Alto
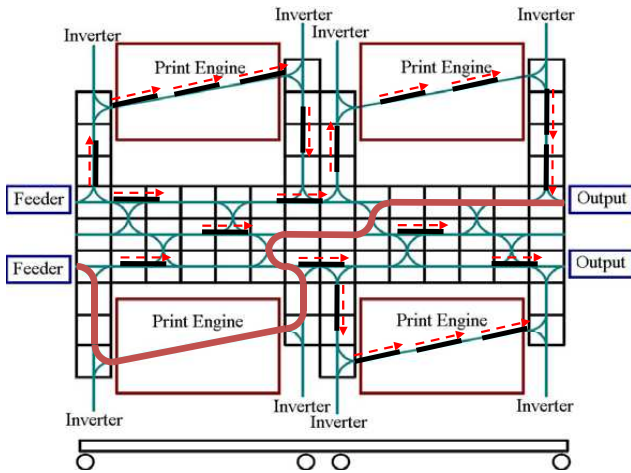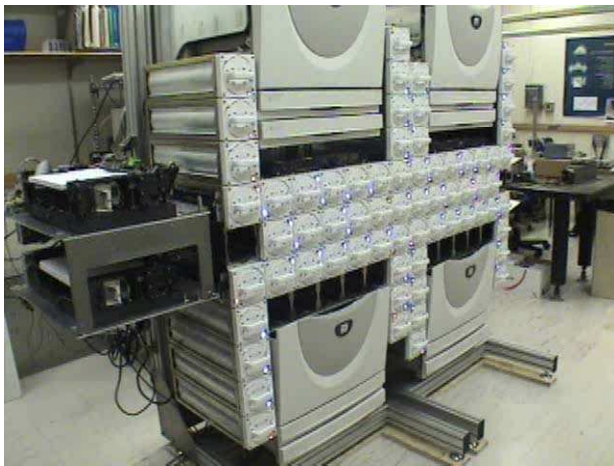
Figure 1: A modular printer and schematic side view showing how the four print engines and approximately 170 other modules are connected. The solid line shows a sample sheet plan together with other in-flight sheets in the system.

Research Center (PARC). It has over 170 independently controlled modules and many possible paper paths linking the paper feeders to the possible output trays. Multiple feeders allow blank sheets to enter the printer at a high rate and multiple finishers allow several jobs to run simultaneously. Having redundant paths through the machine enables graceful degradation of performance when modules fail. By building the system out of relatively small modules, we enable easy reconfiguration of the components to add new modules and functionality. Each module has a limited number of discrete actions it can perform, also known as *capabilities*, and for many of these actions the planner is allowed to control their durations within a range spanning three orders of magnitude.

A printer controller works in an on-line real-time and continual planning environment with three on-going processes: (1) on-line arrival of new goals; (2) planning for known goals; and (3) executions of previously found plans. While usually sequential for any given goal, these processes are interleaved between different goals received. Figure 2 sketches the different steps in the plan life cycle managed by the plan manager.

After the controller issues new plans, there is an additional negotiation protocol before the plan is committed. First, each
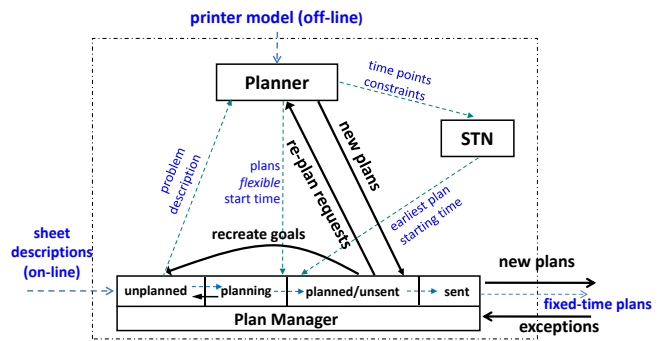

Figure 2: System architecture, showing the steps involved in nominal planning (dashed lines) and replanning (solid lines).

plan step is *proposed* to the module involved. If all involved modules *accept* their proposed actions, then the plan is *committed*. As we discuss below, this commitment means that modules become responsible for notifying the controller if they fail to complete an action or realize that they will not be able to perform a planned action in the future.

Sheet Planning

The sheet planner builds a plan for each sheet of a job using a combination of regression state-space planning and partial-order scheduling. It plans by adding one module action at a time, starting from a finisher until the sequence of actions reaches a feeder. Adding an action to a sheet's itinerary causes resource allocations to be made on any resources required for the execution of that action. Given the media path redundancies in RMP, the planner usually faces multiple choices about which action to add at each planning step. To organize this search, the planner uses best-first A* search with a planning-graph heuristic (Ruml *et al.* 2005; Do and Ruml 2006), adjusted with resource conflicts, that estimates how promising each potential route is. To maintain maximum flexibility, all action times are managed using temporal constraints instead of absolute times. The planner attempts to minimize the earliest time the last action of the currently planned sheet could end, in essence optimizing the system's throughput. The planner uses no domain-dependent search control knowledge, allowing us to use the same planner to run very different printing systems at full productivity.

Our system has been used successfully to control hardware prototypes at PARC (four monochrome engines, giving a total of 220ppm) and at Xerox (two monochrome and two color engines, yielding 180ppm), as well as hundreds of hypothetical RMP systems in simulation, all at their full productivity. These prototypes printers run at speeds higher than any cut-sheet production printers current on the market. We have also built a tool to automatically convert our custom domain language into the PDDL2.1 temporal planning language, allowing us to test current state-of-the-art planners such as LPG (Gerevini *et al.* 2003) and SGPlan (Chen *et al.* 2006), winner of the last two planning competitions. As reported in detail by Do *et al.* (2008), neither of these planners can handle a single sheet for the printer shown in Figure 1. For a much simpler printer, our planner out-performed both LPG and SGPlan by more than 1000x for jobs of up to 15 sheets, which already stretched the limits of LPG and SG-

Plan. For this simpler machine, our planner can plan very quickly for hundreds of sheets easily.

Handling Execution Failures

Imagine a printer or copier that never seems to jam, but just runs a little slower as the month goes on. Once a month, someone opens the covers, removes some jammed sheets, and the system is back at full productivity. The RMP systems that our planner is used to control are designed to fulfill this vision of partial productivity when a subset of the modules are down. To make this transition transparent to the users (and thus increase the perceived reliability of the system), we have been concentrating on developing exception handling techniques that minimize user interventions without stopping or slowing down the machine. Current products perform exception handling using rules hard-coded into each machine module. This technique works well for simple straight-line systems, but would be limited to a small predefined subset of failures in more complex topologies. In our modular RMP systems, there are a countless number of different printer configurations and failure possibilities, so we would prefer a more general exception handling approach.

Since all plans tightly interact through various scheduling and temporal constraints, whether or not they belong to the same jobs, an exception affecting any single plan can affect the executability of other plans and the final job integrity. Plans in different stages of their life cycle need to be analyzed and treated differently (see Figure 2). While *unsent* plans can be canceled, we need new plans for the sheets that are already in-flight at the time exception occurs. In this section, we first discuss the types of exceptions that we can currently handle and how the plan manager reacts to them, we then concentrate on the hardest part of the exception handling framework: finding new set of consistent plans for in-flight sheets.

Basic Exception Handling

Our planner can handle several types of exceptions. When a given exception occurs, the planner will receive a special message from the machine controller in real time. Depending on the failure, the planner does one or more of: adjusting the internal plan queue, updating the machine model and sending new plans to replace the ones that are currently executing. Figure 2 shows in solid lines the possible steps of the replanning process. The dashed lines in this figure show the steps in nominal planning, as described in (Do *et al.* 2008). Next, we discuss in detail each of the different failure scenarios.

Plan Rejection: When a plan is sent to the machine controller to execute, the controller may reject the plan if one of the relevant modules cannot commit to executing its requested action at the time defined by the planner. While such rejections are rare, they can be caused by module constraints that are outside the scope of the planner's model. For example, a binding module may need time to bring its glue reservoir to the proper temperature—a state variable and constraint not currently modeled in our system. When a plan is rejected, the planner will cancel all plans in the *unsent* queue, in addition to the recently sent and rejected plan. All goals corresponding to those plans will be rolled back to the *unplanned* queue. Even plans that are not directly affected by the error message also need to be canceled and rolled back because those plans were made after the commitments had been made for the rejected plan.

Module Update: Machine modules can go *off*-line due to a hardware failure, such as a sheet jam, a benign event, such as running out of paper in a feed tray, or an unmodeled process, such as print engine self-adjustment. Similarly, they can come *on*-line when they are repaired, adjusted, or otherwise made ready. When this happens, the module controller will send message to the planner indicating which of the module's capabilities is now on/off. If a given capability is turned *off*, then the planner will remove the corresponding action from consideration in future planning episodes. If a given capability is turned *on*, then the planner will add it to the action set for future planning episodes.

Break-in-Future: When a module changes the status of some of its capabilities from *on* to *off*, currently executing or unsent plans using that module may become invalid. In this case, the module controller will send messages to the planner indicating which plans are affected. The planner will cancel the affected unsent plans and subsequent plans and move the goals back to the *unplanned* queue. For plans that are executing and thus correspond to sheets that have already been fed into the machine, the planner needs to find new plans for the affected sheets so that they can get to the correct finisher tray without going through the affected modules. The next section describes in detail how to reroute those in-flight sheets.

Broken: This type of exception happens when one or more sheets are jammed in the system. The *broken* messages sent to the planner include the ids of all sheets that are jammed and thus cannot be reused or rerouted because of the failure. When some sheets jam, they normally also disable some modules and thus the *broken* messages normally accompany several *module update* messages, which are described above. The handling of the *broken* exception is similar to the handling of the *break-in-future* exception in many respects: it involves canceling of unsent plans and finding new plans for the in-flight sheets. However, the main differences are: (1) in-flight sheets that were jammed cannot be rerouted; and (2) more critically, the jammed sheets break print-job integrity. We discuss this in detail next.

In-flight Sheet Replanning

In this section, we discuss the problem of finding a new set of plans for in-flight sheets when a sheet is jammed or a module to be used by some plans is broken. The constraints that make replanning more challenging than nominal planning are:

- Sheets cannot stop or slow down while the planner searches for new plans for all in-flight sheets. Thus, if the planner takes too much time to find new plans, the jams and/or module failures will cascade.

- All newly found plans do not have flexible starting times as in the nominal planning case, but should all start from the location at which the sheets are projected to be when the plans are found. The new locations depend on the actual replanning time of the planner.

- The sheets from jobs without jammed sheets still need to finish in the correct finisher tray and in order. Any out-of-order sheets (and all later ones in the same job) should be rerouted to a *purge* tray.
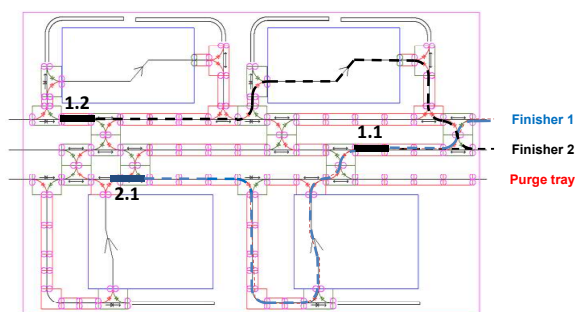
Figure 3: Replanning Example (before jam): sheet 1.1 and 1.2 are planned to enter finisher 2, and sheet 2.1 to finisher 1.
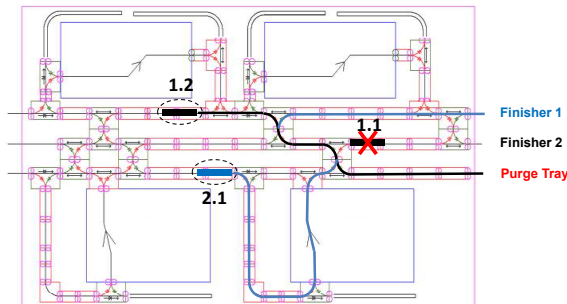

Figure 4: Replanning Example (after jam): sheet 1.1 is jammed, which requires the planner to reroute sheet 1.2 to the purge tray and reroute sheet 2.1 to circumvent the jammed sheet before going to finisher 1.

Replanning involves four main steps: (1) create new goals for the in-flight sheets; (2) predict (an upper bound on) the re-planning time; (3) project the sheets according to the original trajectory and the predicted planning time to find their future locations, which will form the new initial state of the replanning problem; (4) find plans for all sheets that are salvageable (i.e., possible to avoid broken modules and jammed sheets in time), satisfying the constraints listed above.

Example: Here we provide a concrete example illustrating our replanning procedure. Figures 3 & 4 show a scenario in which there are three in-flight sheets: $S_{1.1}$ and $S_{1.2}$ belong to the same job and were planned to go to finisher 2 (in the middle); sheet $S_{2.1}$ belongs to a different job and is scheduled to go to finisher 1. The third finisher is the purge tray. The original routes are indicated by the dashed lines in Figure 3. Assume that $S_{1.1}$ is jammed. According to the original routes, we have: (i) $S_{1.2}$ will arrive in the finisher tray out-of-order (because $S_{1.1}$ did not arrive before it); (ii) $S_{2.1}$ will crash into the module where $S_{1.1}$ jammed. Therefore, we need to find new plans for those two sheets so that $S_{1.2}$ will instead go to the purge tray and $S_{2.1}$ goes around $S_{1.1}$. Finding those plans takes time and given that we cannot stop or slow $S_{1.2}$ and $S_{1.2}$ down while finding the new plans for them, those two sheets will continue their original trajectories to the new locations, which were circled in Figure 4. From there, the printer controller will apply the new plans, which were indicated by the solid lines, that avoid further cascading failures and also guarantee job integrity. After the replanning is done, the planner will regenerate $S_{1.1}$ and $S_{1.2}$ again.

The example above shows one replanning strategy where the new goal for out-of-order sheet $S_{1.2}$ is set to go to the
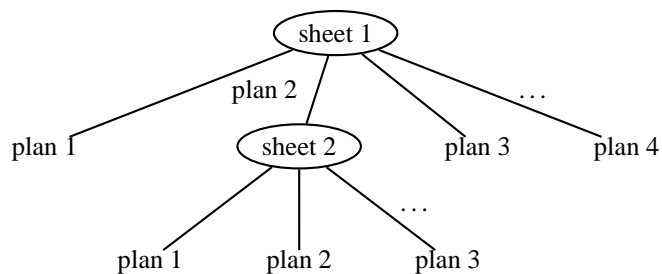

Figure 5: Chaining any searches together gives a search tree with potentially infinite branching factor.

purge tray. This is the default strategy in our replanner that tries to clear out the machine and finish the replanning process as quickly as possible to return to normal operation. However, there are scenarios where printing medias are expensive or confidential and purging them is not desirable. In those scenarios, we have also experimented with a different strategy that does not purge $S_{1.2}$ but keep it in the machine (e.g. loop it) while waiting for $S_{1.1}$ to be reprinted, then $S_{1.2}$ is routed to the original finisher. We have tested this for a small number of sheets, although more sheets could be saved if one is allowed to slow down the transports while looping them.

Chained BFS: For normal operation, the planner uses A* to find the plan for a given sheet that can end soonest, given the (temporally flexible) plans for the previous sheets. A plan always exists if scheduled sufficiently far in the future. For re-routing, the problem is different. We must find jointly feasible plans for as many in-flight sheets as possible. We cannot greedily plan one sheet at a time, committing irrevocably to the plans for all previous sheets, because the plan selected for one sheet might render subsequent sheets infeasible.

We considered two strategies to solve this problem. The first was to simply plan in the joint action space of all sheets. This results in a large branching factor and it was not clear to us how to design an effective heuristic evaluation function. We chose a different approach, in which we can retain the view of planning for each sheet individually using heuristic search. However, we overlay an additional search on top of this, as depicted in Figure 5. In the high-level search, a branching node represents the situation in which we have selected certain specific plans for all previous sheets and it is time to select a plan for an additional sheet. The children of that node represent commitments to the different possible plans for that additional sheet. By considering different paths in the high-level search tree, we can consider different combinations of plans for the different sheets. We call this approach *chained best-first search*. In our current implementation, sheets are replanned in their original order, as an approximation of "distance from exit." An alternative approach is to replan in the order of "urgency" defined as the time left to reroute a sheet before it becomes unsalvageable.

Because the children of a node represent the possible plans returned by a best-first search, the children are not available all at once. Instead, an individual sheet-level planning search will encounter goal nodes one at a time. We cannot terminate the search when we find the first goal node because we have no guarantee that the first goal will make the most subsequent sheets feasible. Finding a goal merely results in an new branch in the high-level space, and to retain complete-

Chained BFS (*problems*)
if *problems* is empty, return success
$p \leftarrow$ remove first problem from *problems*
initialize *openlist* for p
repeat until *openlist* is empty or node limit is reached:
    $n \leftarrow$ best node on *openlist*
    if n is a goal, call ChainedBFS with remaining problems
    expand n, adding any children to *openlist*

Figure 6: Sketch of Chained Best-First Search with a depth-first strategy.

ness we must retain the ability to continue our search and uncover additional possible plans. In fact, in printers such as ours that contain loops in the paper path, there may be an infinite number of possible plans for a given sheet. Fundamentally, the high-level search must explore a tree where nodes are expanded incrementally and the branching factor is potentially infinite.

We identified three possible strategies for searching a tree with infinite branching factor. The first is a best-first approach, in which one formulates a traditional heuristic evaluation function for the high-level nodes. These nodes represent commitments to complete plans for a subset of the in-flight sheets, so the heuristic function needs to estimate the probability that those plans will allow feasible plans for the remaining sheets to be found. The infinite branching factor could be handled using Partial-Expansion A* (Yoshizumi *et al.* 2000), although this would require a non-trivial lower bound on the heuristic value of the plans that have not yet been found. It was not clear to us how this might be done. The second possible strategy we considered was limited discrepancy search (Korf 1996). Unlike depth-first search, limited discrepancy search doesn't necessarily visit all the children of a node, which are potentially infinite for us. The disadvantage to this method is that, because we revisit each node many times with different discrepancy bounds, we will suffer considerable node regeneration overhead.

The third strategy, and the one we used in our implementation, is perhaps the simplest: depth-first search. Figure 6 shows a pseudo-code sketch. Because we have a fixed number of sheets to replan, the high-level search tree has bounded depth. To cope with the potentially infinite branching factor, we impose a limit on the number of nodes each low-level sheet planning search may expand. This avoids the danger of searching forever at one high-level node without finding another goal, and is reminiscent of iterative broadening (Ginsberg and Harvey 1992). To guide the sheet-level planning, we use a heuristic that minimizes plan duration. This attempts to minimize resource use in the machine and maximize the probability that other sheets will have feasible plans.

Multi-Objective Search

Our second major extension to previous work is aimed at better meeting shop owner's needs in the nominal case. Previously, the planner's objective has been to run multi-engine reconfigurable printers at full productivity. Productivity, while very important, is only one of the many optimization criteria that naturally exist in real-world planning and scheduling applications like the printer control domain. In this section, we will describe several additional objective functions that were pointed out as important by our industrial partner, and discuss how we extended the planning framework to handle them.

Optimizing for Printing Cost

For systems with heterogeneous print engines, the cost of printing a given page depends on which of the engines is used. For example, it is costlier to print a black and white page on a color engine than a monochrome one. Thus, to minimize the overall printing cost, one should use the engines with the lowest printing cost that still satisfy the image type and quality requirements of a given print job. By doing so, only a subset of all the available engines will be used for printing a job and thus the overall productivity may be reduced.

To strike a balance between machine productivity and printing cost, we have implemented a multi-objective search framework that tradeoffs productivity for cost and vice versa. Even though the framework only supports these two potentially conflicting objectives at the moment, it can be easily extended to support additional objectives such as minimizing machine physical degradation that we refer to as *wear and tear*. We show that by combining different performance criteria into a single objective, the same optimization framework that works so well for single-objective planning can be efficiently applied to the multi-objective case. Below are the main steps to extending the planner from supporting single objective to multiple objectives.

Step 1: extend the planner's representation of machine capabilities to model action cost. Specifically, we added a cost field representing the cost of executing each capability. In addition, there is an overall objective field with user-supplied weights for each of the two objectives: $obj = \min w_1 * t + w_2 * c$, where t is the end time and c is the accumulated total cost of printing all sheets.

Step 2: heuristic estimation: to find the best route for a given sheet, we estimate how good a potential route is according to each of the objective functions. Finishing time is estimated using temporal planning graph adjusted with resource conflicts between different sheets (Do and Ruml 2006). To estimate the total plan execution cost, we use dynamic programming starting from the initial state (i.e. sheet in the feeder) to compute the total cost to reach different reachable states. The computation is similar to cost propagation on the planning graph as in the Sapa planner (Do and Kambhampati 2002).

Step 3: extend the search algorithm from considering only a single objective to multiple ones. The estimations on total time and cost are combined using the user-supplied weights (as described in Step 1) to compare nodes in the best-first A^* search algorithm. Given that both heuristics for time and cost are admissible, like the single objective planner, our planner guarantees to find optimal solution for any given sheet. Note that if the weights are not given, the planner chooses to prioritize the objectives. For example, the planner can first find the plan that has the lowest cost, and then break ties favoring plans with higher productivity, then favoring one with lower wear and tear, and so on. The new search algorithm has been implemented and fully integrated into our planner. The default option without weights specified is optimizing for productivity and break ties on total cost.

Other Objective Functions: Besides optimizing for speed and cost, we recently extended the multi-objective framework to balance between productivity and diagnostic information gain, the goal of which is to locate one or more failed modules with the fewest test sheets (Kuhn *et al.* 2008). The exceptions we handle are different from the ones discussed in the previous section. Here we assume the failures do not (a) cause the plan to become inexecutable, (b) violate the job integrity, or (c) disable any capabilities. However, these failures will cause incorrect plan output in the form of minor physical damages of the finished sheets. For example, the sheets delivered to the finisher tray might have a small tear on the edge or "dog-eared" corner — physical damage that is small enough to not cause an actual paper jam. We know that the damage was caused by some module that got used by the plan, but we do not know which one. To locate a failed module, we need to find a set of plans, the execution of which will pinpoint the failed module using the fewest number of sheets on average, without sacrificing too much of the overall productivity. Compared to other approaches such as passive and explicit diagnosis, this approach significantly reduces the number of wasted sheets, often by an order of magnitude if the fault is intermittent, the most common failure type in our printer.

Planning for Image Quality Consistency

Maintaining image consistency across a set of heterogeneous engines is especially important for a multi-engine printing system. The planner achieves this by enforcing additional image-consistency constraints while searching for an optimal plan. In color science, the (in)consistency of two colors is measured by a function, often denoted ΔE, that calculates the distance between them in some device-independent color space. While there exist a variety of such functions in the color science literature (the most popular of which is called $\Delta E2000$), for our planning purpose it suffices to assume that given any two engines, a ΔE function returns a non-negative real-valued scalar, called ΔE distance, that measures the discrepancy in *perceived* color as a result of printing the same image on these two engines. Because facing pages (i.e., pages that face each other in a bound book or magazine) are most sensitive to image-consistency issues, we have implemented the following constraints in our planner:

1. *facing-page constraints* that require the facing pages of a job be printed by the same print engine
2. ΔE *constraints* that allow only engines within some maximum ΔE distance to print facing pages

Given that in reality no two engines can have a ΔE distance of zero, the facing-page constraints can be viewed as a special case of the ΔE constraints with the maximum ΔE distance set to zero. Thus, we only need to focus on the latter, which is more general. To enforce ΔE constraints, the planner keeps track of the set of print capabilities that can be used to print the front side of a sheet, which is constrained by the print action applied to the back side of its previous sheet. Since the first sheet of a job does not have a previous sheet, the set of print capabilities eligible for printing its front side is unconstrained (i.e., equal to the entire set of print capabilities). For subsequent sheets of the same job, however, only a subset of print capabilities is allowed. Such a subset is computed based on the ΔE constraints by including only capabilities of those engines whose ΔE distance to the print engine

that printed the back side of the previous sheet is less than or equal to some maximum distance. In most cases, this has to be determined on-line, because the ΔE distance between a pair of engines can drift over time. Thus, our planner maintains an on-line version of a pairwise ΔE-distance matrix for all the engines in a printer.

While adding extra image-consistency constraints can reduce the brute-force search space (if the constraints make the set of reachable states smaller), in practice we found this often makes the search problem *harder*, because the heuristic computed for the unconstrained problem, while still admissible, is no longer informative. To improve the accuracy of the heuristic, the planner computes the temporal planning-graph heuristic for all legal combinations of print capabilities that can be used to print one side of a sheet, and then stores them in multiple lookup tables, one for each combination. When a heuristic estimate for a search node is needed, the planner calculates an index into the lookup table based on the state description (e.g., sheet location, black or color printing), in much the same way how lookups are done in pattern databases (Culberson and Schaeffer 1998). On the implementation side, a hash table of hash tables is used to store multiple lookup tables, but for any given sheet only the relevant hash table(s) is loaded before the sheet is being planned, since the set of eligible print actions is known and fixed at that time.

Since there are only a limited ways of printing a single face of a sheet, this approach to improving heuristic accuracy has little overhead yet can significantly reduce the time it takes to find an itinerary. Interestingly, the same approach can also be used to improve the accuracy of the heuristic under exceptions in which jammed sheets block the media paths to some engines, which force the planner to work with only the set of engines that are unblocked, giving rise to planning problems that are similar to enforcing the ΔE constraints.

Planning with Constrained Action Set

From a pure planning perspective, our approach to planning for image-quality consistency corresponds to solving a constrained planning problem with a reduced set of actions (compared to its unconstrained version). Given a planning problem with k actions, one can create $O(2^k)$ different versions of the constrained problem. Thus, pre-computing the temporal planning-graph heuristic for all possible subsets of actions can quickly become infeasible as k increases. Here we describe a general solution that strikes a balance between heuristic accuracy and the space overhead for storing multiple lookup tables, one for each subset of actions. The idea is to limit m, the maximum number of actions that are removed from the unconstrained problem, and compute heuristic lookup tables only for those constrained problems. For example, it is usually feasible to enumerate those constrained problems in which only one or two actions are removed from the action set. To compute the heuristic value of a state in a constrained problem that is not included in this pre-computed set, the algorithm consults all the lookup tables whose removed actions form a subset of the actions removed in the constrained problem, and returns the maximum value as the heuristic estimate of the state, since the value returned by any of the lookup tables is admissible.

More formally, let $h(s|P)$ be an admissible heuristic estimate for state s in the constrained problem with the set of

actions $P \subseteq A$ removed from the original action set A, and let m be the maximum number of actions removed in any constrained problems for which the heuristic is pre-computed. The heuristic estimate $h(s|P)$ can be calculated as follows,

$$h(s|P) \;=\; \begin{cases} h(s|P) & \text{if } |P| \leq m \\ \max_{Q \subset P \,\wedge\, |Q|=m} \; h(s|Q) & \text{otherwise} \end{cases}$$

The new heuristic resembles the h^m family of admissible heuristics (Haslum and Geffner 2000), where m limits the maximum cardinality of the set of atoms considered in the construction of the heuristic. The difference is that our heuristic considers the set of removed actions, whereas the h^m heuristic considers the set of satisfied atoms. Our heuristic can also be seen as a kind of multiple pattern databases in which one can take the maximum over a set of heuristic estimates without losing admissibility, although ours is based on action-space abstraction and (multiple) pattern databases are based on state-space abstraction.

Experience in Practice

In collaboration with Xerox, we have deployed our planner to control three physical prototype multi-engine printers (one with the schematic view shown in Figure 1). These deployments have been successful and the planner has also been used in simulation to control hundreds of hypothetical printer configurations. The planner is written in Objective Caml, a dialect of ML, and runs on a standard desktop PC under either Linux or Windows. It communicates with the job submitter and the printer controller using ASCII text over sockets. The planner can also communicate with a plan visualizer to graphically display the plans. The shortest single plan for the machine shown in Figure 1 has 25 actions. Given that there are many sheets in the printer at any given time and the planner can plan ahead, the plan manager consistently manages dozens to hundreds of plans. For the most complex machine, the planner consistently on average produces plans within the 0.27 seconds required to keep the printer running at full productivity (220 pages/minute). The ability to use domain-independent planning techniques allows us to use the same planner for very different configurations, without needing any hand-tuned control rules.

Exception Handling: Until now, the exception handling strategies in current production printers have been to: (i) stop the production or (ii) use machine-specific customized local rules to purge sheets in the system. Our work is the first to demonstrate automatic exception handling that does not rely on machine-specific control rules.

The planner can handle the two easiest types of exception: *Plan Reject* and *Module Update* without any difficulties. For the *Brake-In-Future* and *Broken* exceptions, we can currently re-route on the fly up to five sheets for the machine shown in Figure 1 (note that replanning is exponentially harder than nominal planning according the number of in-flight sheets). For the simpler prototype systems at Xerox with fewer (but larger) modules, four print engines, and an aggregate throughput of 180 pages-per-minute, our planner has been able to successfully reroute all reroutable sheets when different jams happen. We have demonstrated our replanning technology in real-time by allowing people come up and either turn on/off modules, or jam sheets intentionally, sometimes right before the sheets hits the broken module. Upon receiving the error messages from the machine controller, the planner is fast enough to reroute the sheets around the failed modules or jammed sheets to the correct locations. Besides experimenting with the physical hardware built at PARC and by Xerox, we have also tested in simulation, by connecting the planner to the visualizer instead of the printer controller. We tried our replanning framework on different hypothetical printer configurations with different fault modes and different exception handling strategies. For example, when the printing media is expensive and the replanning objective function is switched from the default objective function of finish replanning as quickly as possible (which can lead to many purged sheets) to saving as many sheets at possible (which can lead to longer replanning time) then the planner has been able to successfully route up to 2 out-of-order sheets in long routes (that may contain loops) in the system waiting for the jammed sheet to be printed before routed to the correct finisher tray. While achievement of replanning for up to five sheets in a large RMP machine may not seem very impressive, we want to point out that: (1) our planner can reroute all reroutable sheets in simpler machines (which is still much more complex than the biggest multi-engine printer Xerox currently has on the market); (2) the large machine is very complex for automated planning—the last two IPC winners SGPLan and LPG cannot even find plan for a single sheet in nominal planning using the PDDL2.1 version of our printer domain.

Multi-Objective Search: To test the ability to tradeoff between machine productivity and printing cost, we have tested on the model of a four-engine prototype printer built at Xerox. This is a better testbed for the tradeoff investigation because that printer has a mixed set of printer engines (two color and two black-and-white engines) instead of four identical black engines such as in the our system. Moreover, the engines are aligned asymmetrically and thus the paths leading to different engines are slightly different. We have modeled the costs for all different components. We are especially interested in modeling the cost to print black pages on different engines: printing them on more expensive color engines cost more than on cheaper monochrome engines. By varying the weights between the two objective functions, we have been able to show that: (1) increasing the weight given to productivity results in more printer utilization of all four engines; (2) increasing the weight on saving printing cost leads to reductions in the number of unnecessary costly printing, thus fewer black sheets are printed on color engines. We can observe the tradeoff between modules with similar functionality as well, such as between different feeders, finishers, or paper-path. For example, increasing the weight for saving costs lowers the number of sheets fed from a more expensive but faster feeders. We have also tested our multi-objective search on other hypothetical printers with mixed components and similar results were observed. We observed that moving from single to multi-objective search did not slow down our planner and thus does not affect the overall productivity.

We also tested the performance of our planner on image-consistency planning. The model of the printer used has four monochrome engines, two of which are faster but low-quality engines, and the remaining two are slower but high-quality engines. All four engines are connected through asymmetric paper paths. We ran the simulation with a 20-sheet job

that requires using the two high-quality engines for double-sided printing. This can be done with certain ΔE constraints, which can prevent the planner from choosing the two low-quality engines. Since we are particularly interested in the effect of the heuristic on the search performance, we tested the planner with and without using multiple lookup tables, which made a significant difference in the number of node expansions in A* search and planning times. On average, when the multiple lookup table heuristic is used, the planner expands only 1783 nodes per sheet; whereas using the heuristic computed for the unconstrained problem, which grossly underestimates the remaining makespan for constrained problems, needs 6458 node expansions to find a plan. In terms of running time, the one that uses multiple lookup tables is 60% faster than using the naive heuristic.

Conclusions

In this paper, we have described extensions of an online continual planner controlling high-speed modular printer to handle two critical issues: (1) real-time execution failures; and (2) objective functions beyond productivity. We have successfully demonstrated our fast replanning and multi-objective search on three physical prototype printers and many other potential printer configurations in simulation. Our work provides an example of how AI planning and scheduling can find real-world application not just in exotic domains such as spacecraft or mobile robot control, but also for common down-to-earth problems such as printer control. The modular printer domain is representative of a wider class of AI applications that require continual on-line decision-making. Through a novel combination of fast continual temporal planning techniques, we have shown how AI techniques can successfully enable robust, high-performance, autonomous operation without hand-coded control knowledge.

There are other frameworks to handle exceptions and uncertainty in plan execution. Markov decision process (Boutilier *et al.* 1999) and contingency planning (Pryor and Collins 1996) build plans and policies robust to uncertain environment. Planners built on those techniques are normally slow, especially in a real-time dynamic environment with complex temporal constraints like ours. They are not suitable for our domain where exceptions do not happen frequently, but need to be responded to very quickly. Fox *et. al.* (2006) discuss the tradeoff between replanning and plan-repair strategies for handling execution failure. Their algorithms work off-line, instead of in an on-line real-time environment such as ours, and they target different objective function (i.e. plan stability). CASPER system at JPL (Chien *et al.* 1999) uses iterative repairs to continuously modify and update plans to adjust to the dynamic environment. Unlike our system, CASPER uses domain control-rules and thus is less flexible and the replanning decision is also not needed as quickly as in our domain (sub-second).

There are several academic domain-independent planners such as GRT (Refanidis and Vlahavas 2003) and LPG (Gerevini *et al.* 2008) that can optimize for multiple objectives or tradeoff between planning time and plan quality. Standard planning languages, especially PDDL3 (Gerevini and Long 2006), allow specifying complex objective functions in the weighted-sum format (as in our framework). While our planner is also based on domain-independent planning technology and uses an extension of PDDL, our multi-objective planner works in a dynamic online continual environment and interacts with a physical machine, not in an off-line abstracted environment like the mentioned planners.

Currently, we are extending our framework to scale up our real-time replanning framework. While the current planner works for simpler prototype machines (which are still more complex than any multi-engine printer on the market), rerouting all possible sheets for the complex modular printer at PARC is still a challenge. We are also extending to other objective functions such as machine wear and tear.

References

C. Boutilier, T. Dean, and S. Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *JAIR*, 11:1–91, 1999.

Y. Chen, C. Hsu, and B. Wah. Temporal planning using subgoal partitioning and resolution in sgplan. *JAIR*, 26:323–369, 2006.

S. Chien, R. Knight, A. Stechert, R. Sherwood, and G. Rabideau. Using iterative repair to improve the responsiveness of planning and scheduling for autonomous spacecraft. In *Proc. of IJCAI*, 1999.

Joe Culberson and Jonathan Schaeffer. Pattern databases. *Computational Intelligence*, 14(3):318–334, 1998.

Minh B. Do and Subbarao Kambhampati. Sapa: A multi-objective metric temporal planer. *JAIR*, 20:155–194, 2002.

Minh B. Do and Wheeler Ruml. Lessons learned in applying domain-independent planning to high-speed manufacturing. In *Proceedings of ICAPS-06*, pages 370–373, 2006.

Minh Do, Wheeler Ruml, and Rong Zhou. On-line planning and scheduling: An application to controlling modular printers. In *Proceedings of AAAI-08*, 2008.

Maria Fox, Alfonso Gerevini, Derek Long, and Ivan Serina. Plan stability: Replanning versus plan repair. In *Proc. of ICAPS-06*, pages 212–221, 2006.

A. Gerevini and D. Long. Preferences and soft constraints in pddl3. In *Workshop on Preferences and Soft Constraints in Planning, ICAPS06*, 2006.

Alfonso Gerevini, Alessandro Saetti, and Ivan Serina. Planning through stochastic local search and temporal action graphs in lpg. *Journal of Artificial Intelligence Research*, 20:239–290, 2003.

Alfonso Gerevini, Alessandro Saetti, and Ivan Serina. An approach to efficient planning with numerical fluents and multi-criteria plan quality. *Artificial Intelligence*, 2008.

Matthew L. Ginsberg and William D. Harvey. Iterative broadening. *Artificial Intelligence*, 55:367–383, 1992.

P. Haslum and H. Geffner. Admissible heuristics for optimal planning. In *Proceedings of AIPS*, pages 140–149, 2000.

Richard E. Korf. Improved limited discrepancy search. In *Proceedings of AAAI-96*, pages 286–291. MIT Press, 1996.

Lukas Kuhn, Johan de Kleer, Robert Price, Minh Do, and Rong Zhou. Pervasive diagnosis: The integration of active diagnosis into production plans. In *Proceedings of AAAI-08*, 2008.

L. Pryor and G. Collins. Planning for contingencies: A decision-based approach. *JAIR*, 4:287–339, 1996.

Ioanis Refanidis and Ioannis Vlahavas. Multiobjective heuristic state-space planning. *Artificial Intelligence*, 145:1–32, 2003.

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

Takayuki Yoshizumi, Teruhisa Miura, and Toru Ishida. A* with partial expansion for large branching factor problems. In *Proceedings of AAAI-2000*, pages 923–929, 2000.