# A System for Lifelong, Resilient, Job Shop Planning based on Learning Machine Capabilities from Operational Data

**Roni Stern, Wiktor Piotrowski, Lara S. Crawford, Michael Youngblood**

Palo Alto Research Center (PARC)

**Abstract**

We describe the design of a system that *learns* the capabilities of machines in the factory from available operational data, and uses the learned capability model to *plan* how to allocate jobs to machines over time. The described system is designed to be *resilient* to faults and machine performance degradation by including a component that detects such abnormal behaviors, diagnoses their root cause, and adapt the machines' capability models accordingly. These system abilities allow the factory to continue to operate and utilize machines whose performance has been degraded, thereby increasing the factory's overall utilization. In this paper, we describe a reference implementation of such a system based on two publicly available datasets from different factory types: an ion milling factory and a CnC machining factory.

## Introduction

Many production factories rely on a *planner* — either human or automated — to efficiently allocate jobs to machines in order to maximize throughput and factory productivity. That planner is usually faced with a variant of the following setup. The factory consists of a set of *machines*, possibly of different types, which are used to perform a given sequence of *tasks*. To perform a task, the machines in the factory need to perform a set of *operations*. The goal of the planner is to allocate operations to machines that can perform them while aiming to optimize some cost function, e.g., machine utilization.

Many planning algorithms and systems have been developed to solve job shop planning problems. However, these solutions require a human to model the *capabilities* of the machines in the factory, i.e., which operations can be done by which machine and the cost of doing so. Relying on humans to model machine capabilities in a factory has two major limitations. First, modeling by a human is an expensive task that requires sophisticated expertise as well as an understanding of the modeling requirements. Second, the capabilities of a machine may change over time, due to faults and natural degradation, for example, which requires periodic manual "re-modeling" of the factory to account for the changes in the machines' capability models. In general, having an up-to-date capability model of a system is often prohibitively challenging, and is one of the main inhibitors for the wide spread adoption of model-based approaches to automated reasoning tasks such as planning and diagnosis in real factories.



We propose a **holistic system design that either completely avoids or greatly reduces the need for human capability modeling**. In addition, the system we propose **automatically detects and adapt to machine failures and degradations in performance**. Our system includes the following main components:

- **Capability learner.** This component analyzes available operation data from the factory and outputs a capability model for each machine.
- **Planner.** This component uses the given capability models to generate plans and schedules for performing jobs by allocating their constituent operations to suitable machines.
- **Fault detector.** This component tracks the execution of the planned operations and detects when machines fail to execute operations that were allocated to them.
- **Diagnoser.** This component diagnoses failures and outputs potential changes that should be made to the capability models.

Each component in our system is driven by different AI technology. The capability learner applies **data analysis and machine learning techniques** to establish the capabilities of each machine. The planner uses **automated planning techniques** from the AI literature (Ghallab, Nau, and Traverso 2016) to solve planning tasks defined by the given capability model and set of jobs to be performed. Specifically, it compiles the given planning task to a general planning problem defined in the standard Planning Domain Description Language (PDDL) (McDermott et al. 1998), and then uses an off-the-shelf AI PDDL planner to find a solution. The fault detection component can, in general, employ any type of **anomaly detection algorithm** (Chandola, Banerjee, and Kumar 2009) to identify machine failures and degraded performance. The diagnoser component employs **model-based diagnosis techniques**, such as the General Diagnosis Engine (GDE) (De Kleer and Williams 1987) and Conflict-Directed A* (Williams and Ragno 2007), to suggest possible modifications to the capability model that will explain the observed degraded performance.

## Potential Applications

Modeling machine capabilities dynamically using real-time machine data has applications in any manufacturing context that exhibits:

- **Capability redundancy.** This means some operations can be performed by more than a single machine. That is, there are redundancies in the set of machines, equipment, or capabilities, available to perform jobs. Such redundancies can be modeled by our system and leveraged by the planner.
- **Usable degraded-performance machines.** This means some machine fault conditions or health degradations do not necessarily result in the machine having to be shut down or taken out of operation. Rather, such machines may be intelligently managed through factory redundancies and the use of the remaining capabilities of the machine, until a suitable repair can be scheduled.

In addition, the automated method we propose for modeling machine capabilities enables automatic usage analysis and assignment of new jobs to machines, a feature that would be useful in shops that accept a wide variety of customized jobs. Thus, this technology is an enabler for on-demand customization and more flexible local manufacturing.

Concretely, we identify several industries that match these criteria:

- **Subtractive manufacturing (CnC).** CnC machines are often costly to replace and time-consuming to repair. Degraded performance such as more coarse cut accuracy may still be usable to cut some parts but not others, for example. A fault in one axis may still leave the tool able to handle tasks that do not require as many degrees of freedom, as well.
- **Hybrid additive and subtractive manufacturing factories.** While such factories are currently not widespread, they are expected to be suitable applications for our system. This is because hybrid machines inherently include some capability redundancies — some parts can be manufactured either by subtractive or by additive capabilities, while other parts require both capabilities. Thus, even if, e.g., the subtractive tools of a hybrid machine are broken, one could still use that machine for additive tasks.
- **Print shops.** Print shops have a wide variety of custom jobs that come in, and potentially several different types of machines on which they can be run. Many of these machines will have different capabilities such as different finishing options, different print qualities, different paper types, or color vs black and white. If one machine has an issue with a particular paper tray, for example, it can still be used for jobs requiring other sizes or types of paper, and if one machine has an issue with color printing it might still be usable for black and white.
- **Semiconductor manufacturing.** The semiconductor industry has a wide range of manufacturing processes. Our technology is applicable in multiple aspects of these processes. For example, LCD manufacture may involve significant automated material transport from one process to the next. If one of the transport mechanisms were faulted so as to only be able to move more slowly, the schedule of the factory could be modified to take this into account, working around the faulted component where need be.

## Related Work

We and others have explored the concept of using capability models for dynamic planning to enable reconfigurable manufacturing (Crawford et al. 2013; Do et al. 2011). Capability modeling for manufacturing is a challenging topic, however. There has been interest in the standards community in supporting various types of interoperability across different aspects of factory management, in order to enable more flexible factories and more advanced analytics, particularly in machining / subtractive manufacturing (Sprock et al. 2019). Models of machine tool geometry and kinematics exist, though most are in proprietary formats, and there is some effort in tools and standards to allow these models to be shared across platforms (Bärring et al. 2020). There have been attempts to create models of machine capabilities that incorporate a more abstract view and include the capabilities of the controllers as well (Vichare et al. 2009, 2017), but they have not gained general acceptance, nor are we aware of them being used in a practical demonstration. There have also been models generated for specific domains such as robotics (Perzylo et al. 2019). These approaches all require extensive modeling effort by human experts to define and implement the capability models.

Our approach to learning capabilities is inspired by prior work on *learning planning action models* (Yang, Wu, and Jiang 2007; Amir and Chang 2008; Cresswell, McCluskey, and West 2013; Stern and Juba 2017; Aineto, Celorrio, and Onaindia 2019; Juba, Le, and Stern 2021; Juba and Stern 2022; Segura-Muros, Pérez, and Fernández-Olivares 2021). Learning planning action models means learning, usually from observations, a model of the world that enables the application of automated planning algorithms. For example, the FAMA algorithm (Aineto, Celorrio, and Onaindia 2019) accepts a possibly partial view of states and actions observed while executing plans in a domain, and outputs a PDDL action model, which is a model that specifies the preconditions and effects of actions. The LOCM family of algorithms (Cresswell, McCluskey, and West 2013; Gregory and Cresswell 2015) also learns an action model for planning, but assumes as input only sequences of actions. Most prior work on learning action models focused on discrete action models, but recent work has started to explore learning temporal and numeric action models using symbolic regression (Segura-Muros, Pérez, and Fernández-Olivares 2021). Learning machine capablities can be viewed as an application of learning planning action models, where a machine's capabilities translates to a set of actions the machine, specified by their preconditions and effects.

## System Design

Figure 1 illustrates the proposed system, depicted in the block titled "Smart Factories Agent." The proposed system is designed to operate for a given *factory*, which has some form of "clients." The proposed system begins by collecting

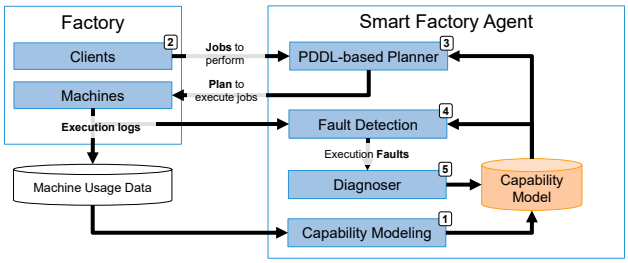


Figure 1: Block diagram of the closed-loop system.

and analyzing usage data from the machines in the relevant factory ("Machine Usage Data" in Figure 1). Then, it runs a learning algorithm (block 1 in Fig. 1) on this data to automatically create models of the capabilities of the machines in the factory. This algorithm may also take into account machine specification information. These capability models represent the abilities and features of the machines in the factory, specifying details about the operations each machine can do. The model captures the abstract information that an automated reasoning engine requires to determine whether a job can be assigned to that machine or not. For example, a capability model may include ranges on physical parameters, such as a range of speeds that the machine can achieve. This capability model is used as follows. When the factory's clients (block 2 in Fig. 1) submit jobs they wish to be performed, our system runs a planning algorithm to automatically create a plan for performing these jobs using the factory's machines. To generate appropriate plans, the planning algorithm needs to know which types of operations each machine can do. To this end, our system uses the capability models we learned from data. In particular, our system compiles the set of jobs it is tasked to perform and the learned capability model into planning problem specified in PDDL. There are many off the shelf algorithms that solve PDDL problems, and our system is designed to use such algorithms (block 3 in Fig. 1) to output a plan that specifies which machines do which jobs and when to do so. Our system also supports generating plans that perform jobs that require multiple machines to be accomplished.

The agent operating the factory accepts the plan our system creates and attempts to execute it.[1] The system continuously monitors the usage data generated by the factory's machines while executing plans, and applies an anomaly detection algorithm to detect execution faults (block 4 in Fig. 1). Since the planner used a learned capability model, a failure may stem from an incorrect assumption about what a machine can do. Alternatively, a fault may stem from a degradation of a machine, which affects the capabilities of that machine. Our system applies a diagnosis algorithm to isolate the root cause of the observed failure, suggesting updates to the capability model that would explain the failure (block 5 in Fig. 1). This allows the system to continue generating plans for the factory and continuously adapt its capability models based on what it observes. Next, we describe the main components of our system, namely, the capability learner, the planner, the fault detector, and the diagnoser, in more detail.

[1]That agent can either be an automated agent or a human operator that observes the proposed plan and acts accordingly.

## Capability Learner

This component accepts as input a machine and operational data collected from it, and outputs a *capability model* for that machine. The capability model of a machine is a model that represents the set of *capabilities* that machine has. An operation can be done by a machine if that machine has the capabilities required to do it. The representation of a *machine capability* depends on the available data and the algorithm used by the capability learner. Below, we specify several possible approaches for implementing the capability learner and the corresponding capability models they learn.

First, we introduce the following notation and terminology. The available operational data, referred to as the training set, is represented as a set of time series observed from each machine. Each element in a time series is an $n$ dimensional vector, whose elements are the values of the observed operational characteristics (OOC) of a machine at the given time. We refer to this vector as an OOC vector. For a time series $T$ we denote by $T[i]$ the $i^{th}$ OOC vector in that time series, and we denote by $T[i][j]$ the $j^{th}$ element of that vector. The elements of an OOC vector may include numeric values such as the rotation speed of a machine. The elements of an OOC vector may also include non-numeric values defining the basic machine features, such as what types of swappable tool tips it can support, or how many axes it has, for CnC machines. Non-numeric OOCs may also indicate something about the mode of operation, such as which tool tip is currently being used.

**Capabilities as Ranges of Values for an OOC** In this implementation of the capability learner, a machine capability is the range of values allowed for one of the OOCs of the machine, e.g., the range of rotation speed values that machine can operate with. The capability model of a machine specifies the allowed range of values for each of the $n$ OOCs of the machine. Correspondingly, the capability learner can learn these values by tracking the min and max values of the OOCs in the training set.

If the allowed values for an OOC do not form a connected interval, the corresponding capability can be modeled as multiple disconnected ranges. The capability learner can learn these ranges by tracking the values of the OOCs in histograms, for example, and then calculating the minimum number of disconnected components required for the model.

**Capabilities as Multi-Dimensional Convex Hulls of OOC Values** In this implementation of the capability learner, a machine capability specifies a *region* of the $n$-dimensional space defined by all possible combination of OOC values. This region specifies the allowed combination of OOC values that machine can support. According to such a capability model, the machine can perform an operation if the OOC values it requires are within one of the regions defined by its capabilities.

One way to represent such a capability model is as a set of multi-dimensional planes that represent a convex area. Correspondingly, the capability learner can learn such an area by computing the *convex hull* of all the OOC vectors seen in the training set. Such a capability learner can employ existing techniques for computing convex hulls of a set of $n$ dimensional vectors. If $n$ is large, i.e., there is a large number of OOCs, it may also be desirable to use dimensionality reduction techniques, such as principle component analysis (PCA) or autoencoders, to reduce the size of the space to be modeled. This may be particularly useful in cases where large numbers of the variables are correlated through physical processes (current and voltage readings, for example). One may even consider simply applying a lower dimensionality projection of the OOC vectors. The resulting low-dimensional representation can then be used in the capability modeling as above. Note that including more OOCs, i.e., higher dimension OOC vectors in the computed capability model may cause learning to be slower but will make the model less conservative.

**Capabilities as Multi-Dimensional Regions of OOC Values** The assumption that the capabilities can be captured as a convex region of the $n$-dimensional space of OOC vectors may not hold. In some cases, the training set may indicate that a better capability model can be obtained by splitting the convex hull of the data in the training set into multiple pieces, each modeled as a separate achievable region. The split version can then be tested with the available data to see whether it now accurately represents the capabilities. One can alternatively employ clustering algorithms to cluster the OOC vectors in the training set in regions of the OOC vector space that are not convex at all, e.g., employing various kernels.

A related approach to capability modeling is to represent the achievable combinations of OOCs as joint distributions. These distributions can be modeled based on observed data, potentially discounted over time, and approximated so as to be represented in a compact form. To determine whether a particular required set of operational characteristics can be executed on a given machine, the probability density at the point corresponding to that combination can be examined. If it is high, then the machine can achieve that operational characteristic. This approach is similar to the modeling approach using histograms above, but using a joint distribution rather than modeling each OOC independently.

Yet another approach is to represent the capabilities of a machine in a neural network classifier, which would be trained to indicate whether a particular set of values (or time series of values) can be achieved on a given machine. This approach does not provide an explicit representation of capabilities, but instead directly maps the desired behavior to yes/no.

**Capabilities as Disjunctions over Non-Numeric OOC Values** The non-numeric data in the training set can be considered in a similar manner, modeling capabilities as a set of acceptable combinations of values. For example, if an operational characteristic can take on values "A", "B", or "C", but only takes on value "A" when a second operational characteristic takes on value "$\alpha$", and otherwise takes on "B" or "C", this set of possible combinations can be represented as a disjunction. The corresponding capability learner can employ known algorithms for learning a disjunctive normal form from data (Mansour 1995). A similar technique of slicing or separating the capability space into multiple regions can be used for this non-numeric data. Models combining numeric and non-numeric data are also possible, of course.

**Capabilities as Patterns in a Time Series** Other machine capability models and learning algorithms are also possible. These may include other ways of modeling multi-variate specific execution patterns that a machine can perform, and then types of tasks it is suitable for. Correspondingly, methods to implement the capability learner may also include applying other temporal data mining techniques to infer complex patterns observed in the data. Because the models are learned based on the observed data, however, if the dataset is limited to a few operations performed by the machine, the model will not capture the full range of the machine capabilities. It is important, therefore, to ensure that the available training data includes a wide range of operations performed on the machine, so as to best capture a machine's full capabilities. This issue can also be mitigated in other ways, such as incorporating machine specifications into the learning algorithm or incorporating human input at critical stages, such as planning for a totally new type of job.

## Planner

This component accepts as input (1) a set of jobs to perform, and (2) a set of available machines, each associated with a set of capabilities the machine is assumed to have. It outputs a *plan*, which is a mapping of operations to machines that will perform them, and a *schedule* specifying when these operations should be executed. The plan must ensure that operations are only done on machines that has appropriate capabilities. The schedule must ensure that the operations needed to create a part are done in the appropriate order.

In our reference implementation, the planner creates a classical planning problem whose solution is a sequence of (operation, machine) pairs. A solution to this problem provides both a plan specifying which operation should be done by which machine as well as the order in which they should be executed. To obtain such a solution, we encode the classical planning problem in PDDL (McDermott et al. 1998), a standard language for classical planning problems, and use an off-the-shelf PDDL planner to solve it. Examples of such planners include Fast Downward (Helmert 2006), Fast Forward (Hoffmann 2001), and Metric FF (Hoffmann 2003). Other implementations of the planner component in our agent may compile the problem to a mixed integer-linear programming (MILP) and use a MILP solver. Explicitly implementing a scheduler or employing a planner that can generate concurrent plans, may also be done.

## Fault Detector

This component accepts as input (1) the plan for performing a set of jobs, i.e., which machine does which operation

in a job; and (2) operation data collected from the system when executing said plan. The output of this component is a value between zero and one that indicates how likely is that the machine is experiencing a fault. There are many ways to implement this component. In our implementation, there is an anomaly detection algorithm that is trained on normal data and then run on the collected operation data. An implementation of such an anomaly detection can be as simple as identifying values of operational characteristics of machines that are inconsistent with our capability model, or only detect faults when an operation of a job does not meet the desired functional (e.g., desired part not manufactured properly) or non-functional requirements (e.g., desired part manufactured too slowly). A different implementation of our fault detector may include more sophisticated anomaly detection algorithms that rely on machine learning algorithms. For example, algorithms such as one-class SVM (Li et al. 2003) and one-class neural networsk such as HRN (Hu et al. 2020). These algorithms train a binary classifier to detect anomalies, and train it by only giving it samples from the normal scenario. In our case, we will train such a classifier on the collected machine data from successful executions. See Pang et al. (Pang et al. 2021) and Ruff et al. (Ruff et al. 2021) for recent surveys on techniques for anomaly detection.

### Diagnoser

This component accepts the same input as the fault detector as well as the capability model learned by the capability learner and the value returned by the fault detection (the estimate for whether or not a fault has occurred). The output of this component is one or more capabilities in the capability model that are expected to be incorrect, correlated with the detected fault. The fault may be due to a physical change in the machine's capabilities or an error in the capability modeling that caused the planner to assign a job to the machine that it was not capable of executing. In either case, the fault signifies a mismatch between the actual and modeled machine capabilities. Our implementation of this component invokes a model-based diagnosis (MBD) algorithm, namely GDE (De Kleer and Williams 1987), which extracts *conflicts* between the assumed capability model and the detected faults, and then identifies *diagnoses* as hitting sets of these conflicts. A conflict in this case is an operation that was not successfully executed by a machine and the capabilities of that machine, according to which that operation should have been executable by that machine. A diagnosis in this case is one or more capabilities in the capability model that may not reflect correctly the capability of the machine. Other implementations of this component could use a variety of different methods, such as a learning-based diagnosis algorithm like the one proposed by Matei et al. (Matei et al. 2020) to diagnose hybrid systems. They used machine learning techniques to general a model of the system that is faster to simulate. Then, a search in the space of possible diagnosis is performed, but more efficiently since simulations are done by the trained networks. Yet another way to implement the diagnosis component is to isolate the faults using parameter tracking algorithms based on optimization algorithms or filtering techniques such as Kalman filers (Welch, Bishop et al. 1995) or particle filters (Gustafsson 2010).

Note that the diagnoser component may associate each diagnosis with a likelihood score that will help selecting which diagnosis to choose. Indeed, many diagnosis algorithms support associating returned diagnosis with some form of score. For example, such a likelihood score can be computed using diagnosis algorithms based on Spectrum-based Fault Localization (SFL), a well-known technique for extracting diagnoses and their likelihoods (Janssen, Abreu, and Van Gemund 2009).

## Implementation

Our first implementation has been demonstrated on two data sets, one on ion milling, from the 2018 PHM Data Challenge (phm 2018), and one on CnC manufacturing, from the NIST Smart Manufacturing Systems (SMS) Testbed (cnc 2021). Ion milling is a process used in semiconductor manufacturing to remove material (also referred to as "etching" in this context). The ion milling data set contains scaled and partially anonymized data from multiple machines, each running processes called "recipes," each of which consists of multiple "recipe steps." There are 24 variables in the data set, most of which are numeric valued. Sample variables include etch beam voltage, etch beam current, and ion gauge pressure. The NIST SMS Testbed contains several CnC manufacturing machines. The data set we have focused on is a raw data set from a Mazak mill-turn machine. This data set contains both symbolic and numeric valued data, including x, y, and z positions, spindle velocities, and status information.

We implemented a simple capability modeler based on ranges as values, as described above. For the ion milling data set, since there are 24 variables, the system learns 24 parameterized capabilities, $x \in [a, b]$. The capabilities are learned from a set of training data simply by observing the range of values present. A sample model is shown in Figure 2. Each recipe step in a recipe has a required range of parameters, so each recipe that the system is requested to execute can be mapped to required ranges for the 24 parameters. These required ranges can then be compared with the capabilities of the different machines to determine which machines can execute the recipe. As described above, we compiled the planning problem of assigning recipes to machines to PDDL (Figure 3), and then used Fast Downward (Helmert 2006), which is a popular off-the-shelf PDDL planner.

In more details, we used the capability model to identify which recipe steps can be performed by which machine. Then, we created a PDDL with an action for every pair of recipe step and machine that can perform it. To ensure recipe steps are done by the same machine and in order, we added to each action preconditions to ensure the previous recipe steps have been performed. Figure 3 shows how an example of the PDDL for these actions.

We simulated the closed-loop system with all the components shown in Figure 1. Incoming jobs were simulated using a reserved test set from the data. The planner assigned the jobs to the multiple machines in the simulated factory. We simulated faults in the factory machines by restricting

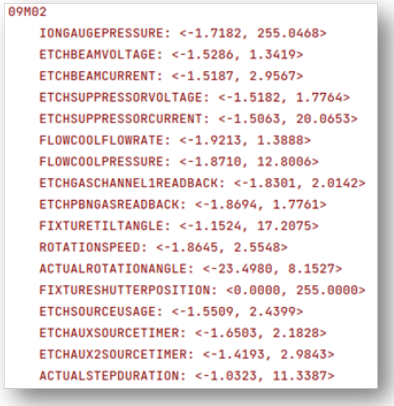


Figure 2: Sample capability model.

```
(:action do_machine_05M02_recipe_149_step_1
    :parameters ()
    :precondition (and (ready))
    :effect (done_machine_05M02_recipe_149_step_1)
)
```

```
(:action do_machine_05M02_recipe_149_step_2
    :parameters ()
    :precondition (and (done_machine_05M02_recipe_149_step_1))
    :effect (done_machine_05M02_recipe_149_step_2)
)
```

```
(:action do_machine_05M02_recipe_149_step_30
    :parameters ()
    :precondition (and (done_machine_05M02_recipe_149_step_29))
    :effect (completed_recipe_149)
)
```

Figure 3: Sample PDDL action models.

the ranges in the machine capabilities, i.e. decreasing $b$ or increasing $a$ in the capability $x \in [a, b]$ in the "true" machine simulation. In this setup, then, a fault was detected if the planner assigned a recipe to a machine with true current capability $x \in [a, b]$, and the recipe required a value outside of $[a, b]$. Once a fault was detection, the system attempted to diagnose the fault, using a GDE-based diagnoser, as described above, to identify which capability (parameter range) needed to be updated, and to restrict that range accordingly. The fault detection and diagnosis triggered replanning, so that the jobs could be reassigned based on the updated capability model.

We also tested the system on the NIST SMS Testbed data. There, we focused on modeling capabilities regarding five numeric values (x position, y position, z position, and two spindle speeds) and two symbolic values (a tool identifier and an execution status). The system performed similarly for this data set.

## Conclusion and Future Work

In this paper, we describe the design of a system that automatically learns a PDDL model of machine capabilities in a factory, and uses the learned model to plan how to allocate tasks to machines by utilizing a domain-independent planner. The system models capabilities of machines by identifying the ranges of valid parameters of machines, or convex models of valid parameter combinations for the machines. Moreover, the system identifies faults when they occur, diagnoses them, and updates its learned capability models and correspondingly updates the resource allocation in real time (reconfiguration). We have implemented a preliminary version of this system on two data sets, from different types of machines — ion milling and CnC. Future work will compare the performance of our system with non-adaptive allocations of tasks, and will evaluate the accuracy of the learned capability models.

## Acknowledgements

This work has been partially funded by the SAIL-ON DARPA program.

## References

2018. PHM Data Challenge 2018. https://phmsociety.org/conference/annual-conference-of-the-phm-society/annual-conference-of-the-prognostics-and-health-management-society-2018-b/phm-data-challenge-6/.

2021. NIST SMS testbed. https://www.nist.gov/laboratories/tools-instruments/smart-manufacturing-systems-sms-test-bed.

Aineto, D.; Celorrio, S.; and Onaindia, E. 2019. Learning action models with minimal observability. *Artificial Intelligence*, 275: 104–137.

Amir, E.; and Chang, A. 2008. Learning partially observable deterministic action models. *Journal of Artificial Intelligence Research*, 33: 349–402.

Bärring, M.; Shao, G.; Helu, M.; and Johansson, B. 2020. A Case Study For Modeling Machine Tool Systems Using Standard Representations. In *2020 ITU Kaleidoscope: Industry-Driven Digital Transformation (ITU K)*, 1–8.

Chandola, V.; Banerjee, A.; and Kumar, V. 2009. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3): 1–58.

Crawford, L. S.; Do, M. B.; Ruml, W. S.; Hindi, H.; Eldershaw, C.; Zhou, R.; Kuhn, L.; Fromherz, M. P. J.; Biegelsen, D.; de Kleer, J.; and Larner, D. 2013. On-Line Reconfigurable Machines. *AI Magazine*, 34(3): 73–88.

Cresswell, S. N.; McCluskey, T. L.; and West, M. M. 2013. Acquiring planning domain models using LOCM. *The Knowledge Engineering Review*, 28(2): 195–213.

De Kleer, J.; and Williams, B. C. 1987. Diagnosing multiple faults. *Artificial intelligence*, 32(1): 97–130.

Do, M.; Okajima, K.; Uckun, S.; Hasegawa, F.; Kawano, Y.; Tanaka, K.; Crawford, L.; Zhang, Y.; and Ohashi, A. 2011. Online Planning for a Material Control System for Liquid Crystal Display Manufacturing.

Ghallab, M.; Nau, D.; and Traverso, P. 2016. *Automated planning and acting*. Cambridge University Press.

Gregory, P.; and Cresswell, S. 2015. Domain Model Acquisition in the Presence of Static Relations in the LOP System. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 97–105.

Gustafsson, F. 2010. Particle filter theory and practice with positioning applications. *IEEE Aerospace and Electronic Systems Magazine*, 25(7): 53–82.

Helmert, M. 2006. The fast downward planning system. *Journal of Artificial Intelligence Research*, 26: 191–246.

Hoffmann, J. 2001. FF: The fast-forward planning system. *AI magazine*, 22(3): 57–57.

Hoffmann, J. 2003. The Metric-FF Planning System: Translating"Ignoring Delete Lists"to Numeric State Variables. *Journal of artificial intelligence research*, 20: 291–341.

Hu, W.; Wang, M.; Qin, Q.; Ma, J.; and Liu, B. 2020. HRN: A Holistic Approach to One Class Learning. In Larochelle, H.; Ranzato, M.; Hadsell, R.; Balcan, M. F.; and Lin, H., eds., *Advances in Neural Information Processing Systems*, volume 33, 19111–19124.

Janssen, T.; Abreu, R.; and Van Gemund, A. J. 2009. Zoltar: a spectrum-based fault localization tool. In *PESEC/FSE workshop on Software integration and evolution*, 23–30.

Juba, B.; Le, H. S.; and Stern, R. 2021. Safe Learning of Lifted Action Models. In *International Conference on Principles of Knowledge Representation and Reasoning (KR)*, 379–389.

Juba, B.; and Stern, R. 2022. Learning Probably Approximately Complete and Safe Action Models for Stochastic Worlds.

Li, K.-L.; Huang, H.-K.; Tian, S.-F.; and Xu, W. 2003. Improving one-class SVM for anomaly detection. In *IEEE International Conference on Machine Learning and Cybernetics*, volume 5, 3077–3081. IEEE.

Mansour, Y. 1995. An O (nlog log n) learning algorithm for DNF under the uniform distribution. *Journal of Computer and System Sciences*, 50(3): 543–550.

Matei, I.; Feldman, A.; de Kleer, J.; and Perez, A. 2020. Real time model-based diagnosis enabled by hybrid modeling. In *Annual Conference of the PHM Society*, volume 12, 10–10.

McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL - the planning domain definition language.

Pang, G.; Shen, C.; Cao, L.; and Hengel, A. V. D. 2021. Deep learning for anomaly detection: A review. *ACM Computing Surveys (CSUR)*, 54(2): 1–38.

Perzylo, A.; Grothoff, J.; Lucio, L.; Weser, M.; Malakuti, S.; Venet, P.; Aravantinos, V.; and Deppe, T. 2019. Capability-based semantic interoperability of manufacturing resources: A BaSys 4.0 perspective. *IFAC-PapersOnLine*, 52(13): 1590–1596. 9th IFAC Conference on Manufacturing Modelling, Management and Control MIM 2019.

Ruff, L.; Kauffmann, J. R.; Vandermeulen, R. A.; Montavon, G.; Samek, W.; Kloft, M.; Dietterich, T. G.; and Müller, K.-R. 2021. A unifying review of deep and shallow anomaly detection. *Proceedings of the IEEE*.

Segura-Muros, J. Á.; Pérez, R.; and Fernández-Olivares, J. 2021. Discovering relational and numerical expressions from plan traces for learning action models. *Applied Intelligence*, 1–17.

Sprock, T.; Sharp, M.; Bernstein, W. Z.; Brundage, M. P.; Helu, M.; and Hedberg, T. 2019. Integrated Operations Management for Distributed Manufacturing. *IFAC-PapersOnLine*, 52(13): 1820–1824. 9th IFAC Conference on Manufacturing Modelling, Management and Control MIM 2019.

Stern, R.; and Juba, B. 2017. Efficient, Safe, and Probably Approximately Complete Learning of Action Models. In *the International Joint Conference on Artificial Intelligence (IJCAI)*, 4405–4411.

Vichare, P.; Nassehi, A.; Kumar, S.; and Newman, S. T. 2009. A Unified Manufacturing Resource Model for representing CNC machining systems. *Robotics and Computer-Integrated Manufacturing*, 25(6): 999–1007. 18th International Conference on Flexible Automation and Intelligent Manufacturing.

Vichare, P.; Zhang, X.; Dhokia, V.; Cheung, W.; Xiao, W.; and Zheng, L. 2017. Computer numerical control machine tool information reusability within virtual machining systems. *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture*, 232.

Welch, G.; Bishop, G.; et al. 1995. An introduction to the Kalman filter.

Williams, B. C.; and Ragno, R. J. 2007. Conflict-directed A* and its role in model-based embedded systems. *Discrete Applied Mathematics*, 155(12): 1562–1595.

Yang, Q.; Wu, K.; and Jiang, Y. 2007. Learning action models from plan examples using weighted MAX-SAT. *Artificial Intelligence*, 171(2-3): 107–143.