A Task Analysis for Automating Arrival Control

RU Technology Report

Jordi Bieger & Kristinn R. Thórisson

Center for Analysis and Design of Intelligent Agents
& School of Computer Science
Reykjavik University

RUTR-CS18001 – May 2018

Reykjavik University – School of Computer Science

Technical Report

INTRODUCTION

This report describes a task analysis for safe, trustworthy automation of *Arrival Control* (AC) – the Air Traffic Control (ATC) job of ensuring that aircraft arrive at the airport with sufficient time in between. This is an implementation of a methodology we have developed for the analysis of tasks, intended to complement the training and education of automatic learners, specifically learners that are intended to be introduced incrementally into automated workflows. Our work specifically targets incremental introduction of *safe and trustworthy automation* for *complex workflows*, hereinafter referred to as STACW. We refer to a chosen task – the main task that is the target of training (Arrival Control in this case) – as the *High-Level Task* (HLT).

In our methodology the HLT is decomposed into a hierarchy of lower-level tasks, which are to be learned cumulatively by an AI system. Arrival control is highly suited to this due to a-priori well-documented structure and ability to be evaluated for quality.

The authors gratefully acknowledge partial support for this work from Isavia - Iceland's aviation authority.

1 Task Analysis Methodology

1.1 Task Evaluation & Selection

The HLT's is analyzed using a *match rubric* (MR), consisting of several criteria on which the high-level task (HLT) must be scored. The MR is intended to give an initial rough indication of 1) how difficult automating a HLT will be and 2) how much it will benefit from our methodology targeting STACW (safe and trustworthy automation for complex workflows), as opposed to other (current and future) methods. Specifically, in cases of relative simplicity, where manual programming is relatively straightforward and no learning is necessary, and where there is no expectation of modification or increase in automation after deployment, the STACW methodology is likely to be overkill. It should be noted that this indication cannot be perfect since perfection would require an actual implementation where its numerous details of execution are settled.

Scoring an HLT on the MR should be done in a collaborative effort between developers, clients and users, and can be refined through a deeper understanding of the STACW methodology, AI system and the HLT itself. As such, the evaluation can be performed in stages: at first rough estimates might be given for each criterion, and if the HLT then looks promising enough (e.g. compared to other candidates) it can be analyzed deeper, which then facilitates further refinement of the estimates, so that a more accurate verdict can be given.

1.2 Task Decomposition

The High-Level Task will need to be decomposed so that components can be cumulatively learned and introduced into the workflow. We recognize three complementary types of decomposition:

Decision-based decomposition identifies all decisions that need to be made in the HLT, at a sufficiently low level. The concept of "decision" is taken very broadly and incorporates for instance: performing actions, making predictions, obtaining particular information, updating the current knowledge base, etc. Lower level decisions may be grouped together into higher level decisions to form a decision hierarchy, where a low-level decision process may be used by multiple higher-level decisions.

Feature-based decomposition (or situation-based decomposition) in a directly-learned task (or decision) attempts to identify (ideally independent) subgroups of features/variables that could be learned separately. For instance, in the "predict arrival time" decision, we have features for wind and precipitation, and we plan to train the system first on "no wind, no precipitation", then on "various wind conditions, no precipitation" and "no wind, various precipitation conditions", and finally on "everything combined". This expect this to lead to faster (curriculum) learning of "everything combined" than if we had started with that from the beginning. Furthermore, by allowing us to "skip" tricky situations, they no longer hold back the introduction of (partial) automation into the workflow; the system could still automate the majority of simpler cases, while warning or deferring to a human operator in trickier ones that have not been adequately learned yet.

Functionality-based decomposition is a decomposition based on the functionality that is introduced into the workflow, which tends to be based mostly on decision-based decompositions and somewhat on feature-based ones. To create and introduce functionality, it is not sufficient that the AI system has (partially) learned the relevant tasks; it is also necessary to integrate such functionality

into the workflow, e.g. by adding certain GUI elements to the workers' software. In addition to being guided by other decompositions, which determine what functionality might be available, this is also guided by the actual workflow and identifying opportunities / situations where automation is most desired.

2 Task Selection for STACW Development

The Match Rubric mentioned above and described in detail in Appendix 1 was adapted from an earlier *Project* Rubric that was meant to determine how well a High-Level Task would be suited for the Project of developing the STACW methodology and testing its feasibility. The criteria for developing and applying the methodology are slightly different: during development we need don't only need to worry about how well STACW will work for the HLT, but also how well the HLT will allow us to test out different approaches for task decomposition, curriculum design, teaching and learning.

Appendix 2 shows how our task analysis methodology was used on four candidate HLTs: Arrival Control, Conflict Resolution, Workload Optimization and Directive Adaptation Assistance. Ultimately, the high-level task of Arrival Control selected for the STACW development project as it was found to best match the desiderata. [1]

In the context of the STACW research project — and perhaps also future application of the project's results — we should initially create one or more simplified abstractions of the selected High-Level Task, with well-specified progressing levels of complexity and realism. This enables us to first test out the feasibility of ideas, without the confounds inherent in a highly complex application domain, with subsequent steps already mapped out to chart the progress of the research.

The process of "mocking up" a simplified task can be performed using several strategies. The main idea is to reduce the number of things that need to be taken into account by the AI system and the teacher. The decompositions mentioned above can help with this. Functionality-based decomposition can be used to select a (small) cluster of subtasks that still form a meaningful unit of functionality. Decision-based decomposition can be used for Wizard of Oz prototyping where (sub)tasks are replaced with components that either provide some arbitrary data or actually hardcode the functionality (so it doesn't need to be learned). Feature-based decomposition is important for the isolation of variables / features / dimensions, which can be omitted or restricted in range — e.g. by discretizing a continuous variable, or only allowing certain (ranges of) values.

For Arrival Control, the main simplification we made is as follows:

There is a fixed number of aircraft, which all have an ID, a velocity and a distance to the runway. An action involves listing an aircraft ID and a plus or minus, signifying that the aircraft should instantly increase/decrease its velocity by 10%. When an aircraft lands, the AI gets a reward of 10, and when another aircraft lands within one minute it gets a punishment of -1000. Subtasks include calculating the time to land, noticing conflicting aircraft pairs, and solving conflicts. The System is provided with

[1] This does not mean that the STACW methodology is unsuited for the other HLTs: they are merely less optimal for the methodology's *development*.

control structures that iterate over: aircraft for which the arrival time is unknown, pairs of aircraft, and the list of potential conflicts.

Variations can then be introduced, such as noise in the observations or actions, partial observability, various kinds of weather, failed/rejected actions, more flexible actions, removal of hardcoded functionality, etc.

Task Characteristics

An important part of task theory is to understand the characteristics of a task-environment.

These characteristics can help compare different tasks, understand how difficult it will be to learn/automate, and pinpoint where the problems lie.

We have identified the following characteristics as relevant when one wants to teach a task to a cumulative learner:

Learnability

The HLT should be learnable by the algorithms and systems we intend to use (i.e. CAPTAIN, but ideally also Hierarchical Reinforcement Learning / Inductive Programming, and perhaps NARS and/or AERA or other similar systems).

'Grading' refers to how well the technologies we're currently considering could potentially learn the task. A 0 here would mean it's impossible, while a 10 would mean it's completely trivial. We should probably aim for something like a 7 or 8, because it still needs to be interesting and be able to derive significant benefit from pedagogy.

One example of something that can reduce learnability is if there is a variable amount of inputs, and evaluating them pairwise cannot be done because there are too many interdependencies.

Teachability

We want to use artificial pedagogy, so the (sub)tasks will need to be teachable by using techniques like part-task training, heuristic rewards, curriculum learning, demonstration, etc. This is more of a meta-requirement that works through other ones like decomposability, data availability, etc.

Grading: This is a function of many criteria and grading guidelines have not yet been developed for the teachability dimension. What needs to be taken into account is how many different teaching methods could potentially be used, both considering the task and the available data. Also, consideration must be made about the necessity and estimated added value of teaching.

Decomposability

The HLT will need to be decomposable into a hierarchy of subtasks. This will be done by humans. The hierarchy is a directed acyclic graph where the nodes are (sub)tasks and the vertices indicate usage of one inside of another. If the hierarchy is a tree, lower-level subtasks are not shared between higher-level ones, which does not allow as much functionality sharing and transfer

learning, but may enable parallelization of learning and could be easier to deal with. Ideally, each subtask should be fairly modular and individually useful for the client/user, and learning one should be facilitated by knowledge of others. If multiple distinct decompositions are possible, this is a plus.

Grading: If the HLT cannot really be decomposed, the grade should be a 0. The ideal case for decomposition arises when the subtask hierarchy is both wide and deep, if subtasks are used by multiple parent/supertasks, if subtasks are modular and individually useful. Here we are primarily concerned with decompositions into subtasks. In other words: temporally abstracted superactions. Decomposition of a task by e.g. considering different regions of input space (e.g. predict X with clear weather, predict X with rain, predict X with wind, etc.) can be useful, but typically not for hierarchical reinforcement learning.

Scalable Complexity

The HLT in its simplest form should be reasonably simple so that it can be the beginning of something larger, as the machine learns more. It also should be complex enough that it can be decomposed into meaningful subtasks. Ideally, we should be able to tune/scale the simplicity/complexity for pedagogical, scientific and practical reasons.

Grading: The ideal case for scalable complexity would be if the simplest case is very simple, and there exists a clear and gradual trajectory through which task complexity/difficulty can be scaled up to an intricate real-world task. Points will be deducted for: difficult simplest case, trivial/uninteresting hardest case, and non-smooth complexity scaling. The last case could occur if there are only a few complexity settings that make sense. The best/smoothest complexity scaling would even allow for using different scales with the same AI instance (i.e. for curriculum learning).

Definability

We should be able to define the HLT with relatively high precision. Most importantly, we need to know exactly what decisions are made—consciously and subconsciously—in terms of (possibly) relevant inputs, type of outputs, and the goal that is served.

Grading: This is a question of how well the HLT can be described in terms of inputs, outputs, methodology and evaluation. Something vague like “prevent planes from crashing into each other based on all available (unspecified) information and expertise” should get an extremely low grade (evaluation is also hard, because no difference is specified between different cases with no crashes). A very high grade means that all of these things need to be specified to a very high level of detail: i.e. a computer needs to know all decisions that need to be made and does not have a human’s common sense, so nothing can be left implicit.

Ease of Performance Evaluation

As far as task analysis is concerned, most tasks in the ATC domain are fairly well defined. However, how to measure performance is not always as clear as would be desired for a learning machine. Ideally we should have an exact and objective formula that calculates the desirability of all actions and outcomes in the HLT.

Grading: This will be 0 if performance evaluation is completely subjective; 10 if it can be captured by a simple algorithm. If this is high then training can be more easily automated; the lower, the more probable the need for a human trainer/teacher.

Data Availability

Data must be available for training. For pure supervised learning, we need input-target pairs where each input specifies the complete situation and each target the current-timestep ideal output. It would in principle be possible to train on these in random order, but if they are part of a sequence we can see if we can do something clever with timing--especially if we also have rewards/goals. If we use a simulator, this type of data will only be available reliably if we already know what to do in every situation, which makes the need for learning a bit redundant, but can be useful in rare situations (e.g. because it allows us to get this knowledge into a different “mind”, which can then perhaps generalize or reuse that knowledge). For reinforcement learning we need rewards or (sub)goals. The more the better. Furthermore, it would be useful to have interesting scenarios as opposed to just random ones. A measure of interestingness would allow us great teaching opportunities. Finally, if data is not available yet, it should at least be easy to acquire.

Grading: Important dimensions for data availability are 1) how much is available, 2) whether it contains all relevant variables, 3) the temporal nature of the data, 4) whether it is easy to work with / extract / adjust, 5) whether it is already available or can be obtained easily, 6) whether data is only from normal/random/boring scenarios or geared towards teaching. Simulators can help generate lots of new data and provide a great sandbox for learning control, especially if success can be quantified and evaluated in real time.

Achievability

The achievability of the potential HLT is a measure that takes into account the status of various other requirements listed below, as well as envisioned time, monetary and manpower budgets for the project.

Grading: The score can be seen as a probability (0 = 0\%, 10 = 100\%) that indicates the likelihood of finishing the project with this HLT given the planned amount of time, resources and manpower.

Task Decomposition

The High-Level Task will need to be decomposed so that components can be cumulatively learned and introduced into the workflow. We recognize three complementary types of decomposition:

Decision-based decomposition identifies all decisions that need to be made in the HLT, at a sufficiently low level. The concept of “decision” is taken very broadly and incorporates for instance: performing actions, making predictions, obtaining particular information, updating the current knowledge base, etc. Lower level decisions may be grouped together into higher level decisions to form a decision hierarchy, where a low-level decision process may be used by multiple higher-level decisions.

Feature-based decomposition (or situation-based decomposition) in a directly-learned task (or decision) attempts to identify (ideally independent) subgroups of features/variables that could be learned separately. For instance, in the "predict arrival time" decision, we have features for wind and precipitation, and we plan to train the system first on "no wind, no precipitation", then on "various wind conditions, no precipitation" and "no wind, various precipitation conditions", and finally on "everything combined". This expect this to lead to faster (curriculum) learning of "everything combined" than if we had started with that from the beginning. Furthermore, by allowing us to "skip" tricky situations, they no longer hold back the introduction of (partial) automation into the workflow; the system could still automate the majority of simpler cases, while warning or deferring to a human operator in trickier ones that have not been adequately learned yet.

Functionality-based decomposition is a decomposition based on the functionality that is introduced into the workflow, which tends to be based mostly on decision-based decompositions and somewhat on feature-based ones. To create and introduce functionality, it is not sufficient that the AI system has (partially) learned the relevant tasks; it is also necessary to integrate such functionality into the workflow, e.g. by adding certain GUI elements to the workers' software. In addition to being guided by other decompositions, which determine what functionality might be available, this is also guided by the actual workflow and identifying opportunities / situations where automation is most desired.

2.1 SIMPLIFIED ARRIVAL CONTROL

Scenarios in Simplified Arrival Control

Scenario S1: Arrival Control

The System is presented with IDs, velocities and distances of a fixed number of aircraft and needs to predict the time at which each aircraft is expected to arrive at each runway (D1.1). Based on this information, the System needs to detect if the arrival times of any two aircraft conflict (D1.2). Detected conflicts must then be resolved by telling an aircraft to speed up or slow down by 10% (D1.3).

Decisions in Simplified Arrival Control

Decision D1: Arrival Control

See Scenario S1.

Input: IDs, velocities and distances of a fixed number of aircraft
Output: ID + speed up/slow down 10% command, or nothing
Method: predict landing times (D1.1), detect conflicts (D1.2), resolve conflicts (D1.3)
Evaluation: +10 per landed aircraft, -1000 per conflict

Decision D1.1: Arrival time prediction

Predict the time at which aircraft A will arrive at the runway.

Input: aircraft info for A (ID, velocity and distance)
Output: time
Method: distance/velocity
Evaluation: ‖predicted time - actual time sans interventions‖$_2$ (L_2 norm)

Decision D1.2: Conflict detection

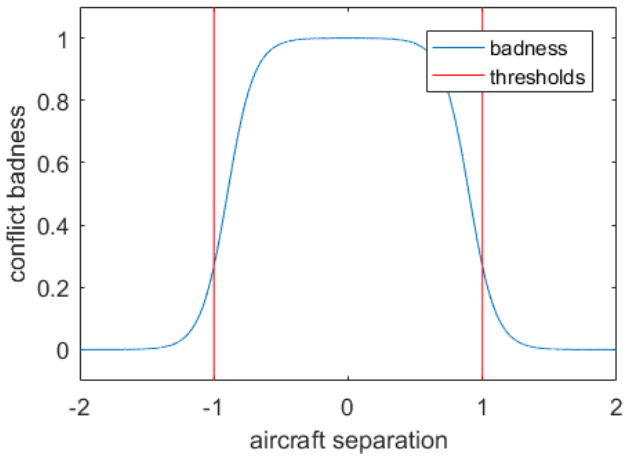
Predict whether two aircraft A and B will have conflicting landing times.

Input: estimated landing times for A and B
Output: yes/no
Method: ‖time$_A$ - time$_B$‖$_1$ < threshold
Evaluation: ‖time$_A$ - time$_B$‖$_1$ < threshold

Decision D1.3: Conflict resolution

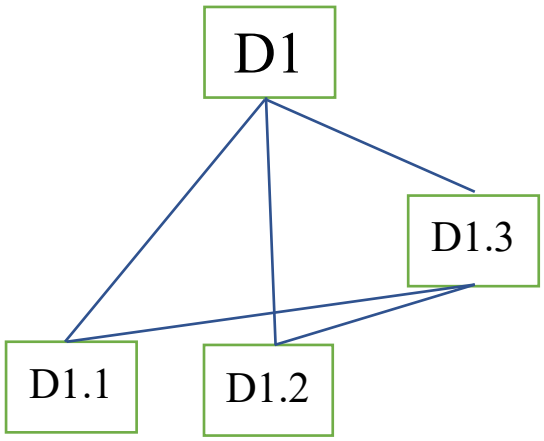
Resolve the conflict between aircraft B and C.

Input: ID, velocity, distance and arrival time of aircraft A, B, C, D, where A is immediately before B, and D immediately after C
Output: ID of B or C + speed up/slow down 10% command, or nothing
Method: See if the conflict can be mitigated by speeding up B, without introducing conflict with A. If not, see if slowing down C works and doesn't conflict with D. If not, do it anyway.
Evaluation: 'global conflict badness after' – 'global conflict badness before' (global conflict badness is the sum of all local conflict badness for aircraft pairs, which should be represented by a smoothed square wave):



Decomposition of Simplified Arrival Control

As described above, decision D1 makes use of D1.1, D1.2 and D1.3, while decision D1.3 also makes use of the functionality of D1.1 and D1.2.



No feature-based or functionality-based decomposition is shown here, because the task is simplified to a point where this would not be instructive. More information about decompositions can be found in section “Task Decomposition” above and Appendix A.

3 AI Training procedures

The optimal way in which training can be conducted depends on the task, the learning system, and available resources for teaching. Machine learning (ML) is often divided into three main paradigms:

- **Supervised learning**, where the AI system is (usually) presented with pairs of inputs and target outputs. In the context of arrival control, this could e.g. require a dataset with ATCO decisions coupled with a rich representation of the information they used. It would be generally assumed that the ATCOs’ decisions were optimal, and the AI would (try to) learn to mimic them.
- **Reinforcement learning**, where the AI system is given information about the environment as well as a desirability score for environment states and actions. The most common way to do reinforcement learning has the AI system make its own decisions in a (simulated) environment and requires humans to code an automatic scoring function. Learned behavior would try to optimize the scoring function, which could lead to superhuman performance.
- **Unsupervised learning**, where the AI system is only given unlabeled inputs/observations. Without a sense of what is right or wrong, this cannot directly result in desirable behavior, but it is sometimes used in combination with other paradigms. By capturing the structure of and regularities in the underlying data distribution, supervised learning can be improved through “pretraining” or “semi-supervised learning”, and a model of environment dynamics can be learned to aid reinforcement learning. Probably the most notable use in arrival control would be to use continuous data from real life or the simulator to teach the AI to predict future environment states.

Ideally we should have an AI system capable of all three methods of learning, to be used depending on the task and availability of data.

The typical machine learning process involves gathering a lot of data, or building a simulator, and then simply giving it to the AI system to train on. From a teaching perspective this is very minimal and suboptimal. There are many teaching strategies that are potentially beneficial (Bieger et al., 2014):

- *Heuristic rewarding* for reinforcement learning, where the AI system does not just receive rewards when an ultimate goal is actually reached, but also received intermediate feedback to guide it in the right direction. In a chess game we only care about win, draw or lose, but in addition to giving large rewards when the game ends (e.g. 1000, 0 or -1000), we could also give small rewards when pieces are captured to indicate that this is a good thing to try/avoid.
- *Shaping*, where a task is first simplified and then gradually increased in difficulty. An important concept in learning theory is Vygotsky's zone of proximal development, which indicates the "zone" just outside the learner's current knowledge/capabilities (so we don't waste time learning what is already known) but not too far away (so learner is not totally confused, overwhelmed and demotivated). This can be accomplished by first training on samples / situations that are easier, such as when predicting aircraft arrival times without wind or precipitation.
- *Part-task training*, where the learner first learns individual parts of a larger whole composite task. Starting from scratch, the whole task may be too far outside the zone of proximal development. Furthermore, there is often a combinatorial explosion of possibilities when decisions in the subtasks need to be made based on different, independent information that can be avoided in this way. This is one reason why task decomposition is a core part of STACW.

There are other teaching methods, including situation selection, demonstration, teleoperation, coaching, explanation and cooperation, which we will not explicitly test in this project.

4 AI Evaluation Procedures

Most AI projects are evaluated by defining a metric and either testing on a separate test set (for supervised learning), or testing in a (slightly) different environment (for reinforcement learning, and future predicting unsupervised learning). We will follow this approach in this project. In our case this will be a numeric function that assigns different (negative) scores to aircraft crashing into each other, aircraft violating separation bounds, fuel costs, delays number of interactions, etc. Ideally this would mirror how such negative events are viewed when resulting from the actions of human ATCOs. This will allow us to evaluate the AI system not only in various simulations but also in light of real-world deployment, where it can be compared with human ATCO performance. Special consideration is needed for the system indicating that it doesn't know what to do (e.g. the test may need to incorporate an ATCO who can take over in this case). Additionally, since it may be impossible for ATCOs to capture everything important in such an evaluation function, it may be necessary to survey ATCOs on how good (and possibly humanlike) they subjectively think the AI system performs.

However, such (a) raw performance is not all that matters (Thórisson et al., 2015; Bieger et al., 2016). In the context of STACW we are especially interested in how (b) *trustworthy* the system is perceived to be. This could again be evaluated *by ATCOs observing the system* passively. Of course it will be more informative if the ATCOs (c) base their judgment on actual experience of working in

tandem with the developed (and possibly develop*ing*) AI system, having observed it in action alongside their own work. In this case we would need to develop some measure of work-load reduction that the AI system enables. We would also want to (d) evaluate the *understandability* of the AI's decisions (Bieger et al., 2017). However, at this point no concrete methods for doing so have been developed, although some ideas can be gleaned from Bieger & Thórisson (2017).

For the development of STACW it is furthermore important to be able to evaluate different teaching methods in tandem with the developed AI system(s). Here we are interested not only in eventual performance, but also in (e) learning speed, (f) data efficiency, (g) generalizability of knowledge/capabilities and flexibility to change (such as if the optimal distance between aircraft changes, or if we want to increase the difficulty/realism of a task compared to a simplified version), and last but not least, (h) knowledge transfer.

So far we have developed evaluation procedures for (i) stand-alone performance, (ii) learn rate and data efficiency for the simplified mockup task, and are capable of (iii) qualitatively estimating understandability and flexibility of the AI system and its decisions. Concrete evaluation procedures for more realistic settings still need to be selected / developed, based on discussions with domain experts and users as well as the exact details of the tasks and sub-tasks and chosen training procedures.

5 AI Requirements

We have analyzed the requirements for an *ideal AI system that the STACW goals imply*. A compact summary on these follows.

Safety-critical and complex workflows should be *minimally disrupted* by automation introduction, avoiding large, sudden changes, which means automation must be introduced into the process *gradually*. We ought to start by introducing functionality to automate or assist with a *few small tasks*, and gradually increase the number and scope of automated tasks. This implies a kind of **modularity** and **gradual learning** in the AI system. Furthermore, we want to be able to expand the system's functionality *efficiently* and *without deteriorating performance on older tasks*, which means the system needs to be able to do **robust cumulative learning** (related concepts include "transfer learning" and avoiding "catastrophic forgetting"). Sometimes newer tasks make use of, combine, or subsume other tasks, so the system needs to support **hierarchical** task representation and learning structure. Furthermore, since parameters of tasks in ATC can sometimes change, we need an AI that is **efficiently adaptive** (i.e. the adaptation must be pragmatic and fall within sensible values of time and energy/resource requirements). The holy grail of efficient growth and adaptation would be if it could be done entirely through training/teaching without any manual code changes (**natural growth**). Since we intend to use teaching, the system should also be **teachable** in various ways.

Trustworthiness and safety are enhanced when both the human operator and AI system know the AI system's limits. On the AI side, we want the system to be **robust**, which ideally means it will generalize good decisions to unfamiliar conditions, but at least it means it needs to be **aware of its limitations** and be able to signal them to its superior (which for now is always a human). On the human side, trustworthiness and safety are enabled by experience with the system, **interpretability** of the AI's models and decisions, and/or **traceability** of smaller decisions on (possibly opaque) human-understandable subtasks leading to a decision. Trust can further be earned through learning

quickly from explicit corrections of a human operator using **online one-shot learning**, and through formally verifying components for certification, which requires them to be fairly **granular**.

In the chosen domain of the Air Traffic Control task of Arrival Control (and also conflict resolution), there are further requirements that extend beyond current state-of-the-art machine learning and AI methods. Among the larger ones is the variable number of aircraft. Since the number of aircraft can change at any moment, the AI needs to deal with **variable input sizes**. Furthermore, the ratio of observations to required actions and actual "rewards" is very large (**sparse actions and rewards**), as the time for a typical arrival extends over at least 30 minutes, there are highly **variable delays** between actions and rewards, and there are **complex causal chains** in the relationship between actions and rewards (e.g. a reward given when an aircraft lands successfully does not necessarily imply anything about the latest action that was taken, or in fact any action at all; a punishment given due to a separation conflict may similarly not reflect on the latest action, but rather on the fact that no (good) action was taken in the timesteps since it was predictable). Most contemporary methods are simply incapable of dealing with variable input sizes, and require that the input representation is hacked in some way to make it have a fixed size, which places fundamental limits on the maximum size that can be handled. Furthermore, this complication as well as the others typically results in significantly decreased performance of learning methods that were not explicitly designed to deal with them.

For each individual subtask there may be slightly different requirements based on the nature of the task and the availability of data and teaching resources/knowledge. It is in principle possible — to some degree — to use a different stand-alone algorithm for each subtask, but maintaining other properties like cumulative learning ability, self-awareness of limitations, and interpretability makes this difficult. For numerous reasons this is an inferior approach; a unified system for learning (near) everything to a system that must be engineered and designed bit by bit is much preferable. Such a system is a cornerstone of the present work.

7 AI Candidate Technologies

Below we evaluate a number of well-known and/or promising techniques for meeting these requirements:

- (deep) neural networks ((D)NNs)
- (hierarchical) reinforcement learning (RL and HRL)
- the X classifier system (XCS)
- automatic program learning (APL)
- the Non-Axiomatic Reasoning System (NARS)
- the Autocatalytic Endogenous Reflective Architecture (AERA)

Neural Networks

Neural networks (NNs) especially of the *deep* variety (DNNs) with "deep learning" constitute one of the most popular families of machine learning algorithms today (Goodfellow et al., 2016). These algorithms have somewhat recently facilitated great strides in performance on messy problems like

human perception, language processing and game playing. While they can often attain high levels of performance, they fail on virtually every STACW requirement.

While (D)NNs constitute a large family of slightly different approaches, they all tend to work using immense networks of nodes ("neurons") with weighted connections between them. "Activation" spreads through the network from the input nodes to the output nodes, and if feedback is available the error is "back propagated" to adjust the connection weights so that next time the network will make a smaller error. With a lot of training data consisting of input-output examples, huge deep nets can learn surprisingly complex tasks.

However, the activation of individual "hidden" nodes is meaningless to humans, and the activation of many thousands is inscrutable. Neural networks are therefor often regarded as "black boxes" where we have no idea what they're basing their decisions on. One DNN can learn a complex task, but there is no way to gradually introduce subtasks into the workflow or to adapt just one part. They must go over large data sets multiple times, so online one-shot learning is out of the question, and if one task is learned after another the first will be forgotten.

	supervised learning	unsupervised learning	reinforcement learning	modular	cumulative learning	hierarchical / transfer	adaptive	natural growth	teachable	robust	aware of limitations	legible	traceable	online	one-shot	handles time	granular	variable i/o	variable reward delay	reward/action sparsity	complex causal chains
(D)NN	✓	±	±	✗	✗	✗	✗	✗	±	✗	✗	✗	✗	✗	✗	✗	✗	✗	?	?	?
RL	±	✗	±	✗	✗	✗	✗	✗	±	✗	✗	✗	✗	✓	✗	✗	✗	✗	±	?	?
HRL	±	✗	✓	✓	±	✓	±	✗	±	✗	✗	✗	±	✓	✗	✗	±	✗	±	?	?
(H)XCS	✓	±	✓	✓	?	±	±	✗	±	✗	±	✓	±	±	?	✗	✓	✗	✗	✗	?
IP	✓	±	✓	✓	✗	✓	±	✗	±	✗	✗	✓	✓	±	±	✗	±	±	?	?	?
DT	✓	±	±	±	±	±	±	✗	±	✗	✗	✓	✓	±	±	✗	✗	✗	?	?	?
NARS	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	±	✓	✓	✓	✓	✓	✓	✓	✓
AERA	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	±	✓	✓	✓	✓	✓	✓	✓	✓

Reinforcement Learning

Typical reinforcement learning (RL) algorithms learn online to control some process from (usually) delayed rewards and punishments (Sutton & Barto, 1998). Unlike most supervised learning algorithms, they don't need to know exactly what they should have done in each moment in order to learn.

Non-hierarchical versions of RL must however take the entire state space into account, which is often infeasible. Function approximation (e.g. using other methods like NNs) can somewhat alleviate the issue. However, the learned policy is monolithic and inscrutable to humans, resulting in similar problems that occur with (D)NNs (except raw RL methods have a worse track record w.r.t. performance).

Hierarchical Reinforcement Learning

In reinforcement learning the AI learns to accomplish a task by receiving feedback about the desirability of states of affairs, actions, or combinations thereof, as they occur (Hengst, 2012). In hierarchical reinforcement learning the task is decomposed into a hierarchy of (ideally) smaller tasks. A child task is often viewed as a temporally extended action in the parent task. In the ideal case, this allows for the reuse of subtasks (so they only need to be learned once) and decreases the state*action space for each smaller task.

HRL can work if your entire Task consists of reinforcement learning tasks, but less well when you also have (un)supervised learning tasks. It works especially well if a subtask is used multiple times in different states of the parent task.

Unfortunately, structure is very difficult to learn, so it must typically be specified up front. Like other RL methods, the individual components are typically not interpretable, but if the component tasks are human-understandable (which is likely if we specified them), then a trace of invoked subtasks can make high-level decisions somewhat understandable. Like with other methods, there is typically no way to deal with a variable number of aircraft.

X Classifier System

The X Classifier System (XCS) is a Learning Classifier System (LCS) that learns to act by evolving and tuning a population of classification rules (Wilson, 1995). Over time, the algorithm drives towards a minimal, fit, non-overlapping population of rules. Each rule applies in a certain situation, "proposes" an action and predicts the expected payoff of that action (direct + expectation for best subsequent actions).

The rules are fairly interpretable, making the system as a whole transparent.

XCS works best in classification settings where rewards are immediate and subsequent input-action-reward interactions are independent from each other. Getting it to work with sequences of interactions and (extensively) delayed rewards is tricky.

Hierarchical XCS

Hierarchical XCS (HXCS) works by creating a partitioning of input variables and having different (H)XCS agents pay attention to each (Barry, 2001). This is essentially feature-based decomposition as mentioned above. The main downside is that it only seems to work if the partitions are independent of each other: i.e. if exactly one of the partitions is relevant to the solution.

Decision-based decomposition does not seem entirely impossible, although it requires modifications of HXCS to deal with heterogenous sub-agents. This is likely to lead to problems, since only one sub-agent gets to make a decision at each instant, and if they use different confidence metrics, it will be difficult to decide between them and avoid some agents being “starved“.

Inductive Programming

Inductive programming (IP) is a learning technique where (computer) programs are inferred from experience/data, typically using methods resembling induction or abduction (Flener & Schmid, 2008). By contrast, “deductive programming” would synthesize a program from a specification. While most learning paradigms are more prone to produce relatively simple pattern matching, IP produces a potentially recursive algorithm that can more easily capture a process. IP is usually used

to produce declarative logic programs (ILP) or functional programs (IFP), and not typically to create imperative programs which probably fit more naturally to the AC domain.

The synthesized programs are interpretable, can deal with variable inputs, and can potentially form a hierarchy of reusable programs which could be adapted separately. Current approaches are not really capable of natural growth, cumulative learning, or dealing with unknown situations. Furthermore, it is typically necessary for a programmer to provide a large portion of the program, because it's only feasible to fill in some minor details through learning.

Decision Trees

Decision Trees (DTs) are classification systems that are formed by a tree of (often binary) rules or questions (Kotsiantis, 2013). For a particular input, we start with an hypothesis that it might belong to any input class. As we descend through the DT and answer its questions, we eventually have enough information to (hopefully) classify the input. Algorithms like ID3 and C4.5 can be used to generate DTs through supervised learning, or in other words: to learn what questions should be asked.

The main strength of DTs is their understandability. They also have some limited ability to do online, cumulative and one-shot learning, although this may result in very sub-optimal decision trees. There are also algorithms that allow reinforcement learning to be used. In one sense, DTs are obviously hierarchical, but learning algorithms don't tend to make use of this for transfer learning between branches. Similarly, there is very little meaningful modularity.

NARS

The Non-Axiomatic Reasoning System (NARS) aims to be a general-purpose intelligent system that learns from experience and adapts to unknown environments (Wang, 2013). It is built from the ground up around the Assumption of Insufficient Knowledge & Resources. The multi-layered non-axiomatic logic allows the system to model itself and its own knowledge and limitations.

NARS can grow and expand functionality naturally without re-programming by learning cumulatively, it should be robust to changes in the environment and—given human-understandable inputs—the models are in principle interpretable. The system can provide a trace of activity leading to decisions, and the only potential obstacle to full understandability is that the reasoning traces can become too large to easily comprehend with the limited human brain. The system can learn on-line and occasionally in one shot, it is robust to changes and self-aware of its limitations, and importantly it can easily deal with variable numbers of inputs.

We believe NARS indeed meets—or can be adapted to meet—all of the STACW requirements. The main issue with using it so far has been that NARS is a highly complex, perhaps *overly* general-purpose learning and reasoning system, which is difficult to get started with. We estimate that it would take about one year for a graduate student to sufficiently familiarize themselves with the NARS theory and implementation to create a version that would work for this project, assuming extensive help or collaboration from someone intimately familiar with NARS.

AERA

The Autocatalytic Endogenous Reflective Architecture (AERA) also aims to be a fully general control architecture (Nivel et al., 2013). It creates small models of itself and the environment from experience, that can chain together to reach complex decisions and behaviors. AERA was built with

AIKR in mind and specializes in finding causal structure and taking into account the temporal nature of the domain, making it well-suited to control tasks with resource constraints. AERA's small models are interpretable and decisions can be traced, although there may be too many to easily comprehend in some cases.

Issues with AERA are similar to NARS. In addition, the system is not nearly as well documented, is highly complex and difficult to use. While not impossible, since the original development team has left RU, getting the system running for STACW would likely take 24 man months or more.

Appendix A: Task Selection & Analysis Methodology

The STACW methodology consists of multiple components, including the selection and analysis of a High-Level Task (HLT), selection/design/implementation of an AI system, and guidelines for how to teach, test and interact with it. This section focuses on evaluating the match between a HLT and (the rest of) the STACW methodology. This can help select a HLT for the STACW development project, and help predict the potential difficulties of using STACW on an envisioned HLT in the future.

Determining the match starts with an attempt to fill out the match rubric. In the case of task selection for the STACW project, we have done this for four prospective HLTs. While a (partially) filled out rubric does not tell us everything about the match between STACW and an HLT, it does provide a solid indication of which HLTs are more promising.

For the more promising HLTs, we can then perform a further analysis by forming a decomposition strategy. By decomposing the HLT into a hierarchy of subtasks, and specifying in detail what we know about them and what data we have, more information is gained about the difficulty of automating the HLT (which allows for making better / more confident updates to the rubric). Furthermore, this analysis will serve as the basis for forming a teaching strategy later on.

Task Requirements & Desiderata

While some co-dependence between the below requirements is unavoidable, we have tried to separate them clearly.

1. **Achievability**
    - The achievability of the potential HLT is a measure that takes into account the status of various other requirements listed below, as well as envisioned time, monetary and manpower budgets for the project.
2. **Learnability**
    - The HLT should be learnable by the algorithms and systems we intend to use (i.e. CAPTAIN, but ideally also Hierarchical Reinforcement Learning / Inductive Programming, and perhaps NARS and/or AERA or other similar systems).
3. **Teachability**
    - Since we want to use artificial pedagogy, the (sub)tasks will need to be teachable by using techniques like part-task training, heuristic rewards, curriculum learning, demonstration, etc. This is more of a meta-requirement that works through other ones like decomposability, data availability, etc.
4. **Decomposability**
    - The HLT will need to be decomposable into a hierarchy of subtasks. This will be done by humans. The hierarchy should be a directed acyclic graph where the nodes are (sub)tasks and the vertices indicate usage of one inside of another. If the hierarchy is a tree, lower-level subtasks are not shared between higher-level ones, which does not allow as much functionality sharing and transfer learning, but may enable parallelisation of learning and could be easier to deal with. Ideally, each subtask

should be fairly modular and individually useful for the client/user, and learning one should be facilitated by knowledge of others. If multiple distinct decompositions are possible, this is a plus.

5. **Scalable complexity**
    - The HLT in its simplest form should be reasonably simple so that it can be the beginning of something larger, as the machine learns more. It also should be complex enough that it can be decomposed into meaningful subtasks. Ideally, we should be able to tune/scale the simplicity/complexity for pedagogical, scientific and practical reasons.
6. **Definability**
    - We should be able to define the HLT with relatively high precision. Most importantly, we need to know exactly what decisions are made—consciously and subconsciously—in terms of (possibly) relevant inputs, type of outputs, and the goal that is served.
7. **Ease of performance evaluation**
    - As far as task analysis is concerned, most tasks in the ATC domain are fairly well defined. However, how to measure performance is not always as clear as would be desired for a learning machine. Ideally we should have an exact and objective formula that calculates the desirability of all actions and outcomes in the HLT.
8. **Data availability**
    - We will need data to train on. For pure supervised learning, we need input-target pairs where each input specifies the complete situation and each target the current-timestep ideal output. It would in principle be possible to train on these in random order, but if they are part of a sequence we can see if we can do something clever with timing--especially if we also have rewards/goals. If we use a simulator, this type of data will only be available reliably if we already know what to do in every situation, which makes the need for learning a bit redundant, but can be useful in rare situations (e.g. because it allows us to get this knowledge into a different "mind", which can then perhaps generalize or reuse that knowledge). For reinforcement learning we need rewards or (sub)goals. The more the better. Furthermore, it would be useful to have interesting scenarios as opposed to just random ones. A measure of interestingness would allow us great teaching opportunities. Finally, if data is not available yet, it should at least be easy to acquire.
9. **Added value from automation via machine learning**
    - Not everything can benefit maximally from machine learning. When a task can be defined very precisely, it's often possible to just write a program for it. In other cases, search or planning can also be preferable. This typically requires a model of the world (which could be learned). Learning typically works best when there are a medium amount of relevant inputs that are combined in a somewhat intuitive but not completely clear manner to arrive at a decision whose success can be measured in some way.
10. **Interest to client**
    - The chosen HLT should have some interest for the client in question (e.g. air traffic authorities).

Grading guidelines

For each potential task, the following rubric should be filled out. “Grade” should be a numeric score out of 10, and “Conf” is short for “confidence” and can be a kind of standard deviation. Below each criterion is room for analysis and justification of the grading. Finally, there is an overall verdict. In this rubric we will now use the analysis space to provide guidelines for grading that part.

Achievability	**Grade:**	**Conf:**
The score can be seen as a probability (0 = 0%, 10 = 100%) that indicates the likelihood of finishing the project with this HLT given the planned amount of time, resources and manpower.		
Learnability	**Grade:**	**Conf:**
This is how well the technologies we’re currently considering could potentially learn the task. A 0 here would mean it’s impossible, while a 10 would mean it’s completely trivial. We should probably aim for something like a 7 or 8, because it still needs to be interesting and be able to derive significant benefit from pedagogy. One example of something that can reduce learnability is if there is a variable amount of inputs, and evaluating them pairwise cannot be done because there are too many interdependencies.		
Teachability	**Grade:**	**Conf:**
Since this is a function of many other criteria, I’m not sure how to provide grading guidelines. What needs to be taken into account is how many different teaching methods could potentially be used, both considering the task and the available data. Also, we need to consider the necessity and estimated added value of teaching.		
Decomposability	**Grade:**	**Conf:**
If the HLT cannot really be decomposed, the grade should be a 0. The ideal case for decomposition arises when the subtask hierarchy is both wide and deep, if subtasks are used by multiple parent/supertasks, if subtasks are modular and individually useful. Here we are primarily concerned with decompositions into subtasks. In other words: temporally abstracted superactions. Decomposition of a task by e.g. considering different regions of input space (e.g. predict X with clear weather, predict X with rain, predict X with wind, etc.) can be useful, but typically not for hierarchical reinforcement learning.		
Scalable complexity	**Grade:**	**Conf:**
The ideal case for scalable complexity would be if the simplest case is very simple, and there exists a clear and gradual trajectory through which task complexity/difficulty can be scaled up to an intricate real-world task. Points will be deducted for: difficult simplest case, trivial/uninteresting hardest case, and non-smooth complexity scaling. The last case could occur if there are only a few complexity settings that make sense. The best/smoothest complexity scaling would even allow for using different scales with the same AI instance (i.e. for curriculum learning).		
Definability	**Grade:**	**Conf:**
This is a question of how well the HLT can be described in terms of inputs, outputs, methodology and evaluation. Something vague like “prevent planes from crashing into each other based on all available (unspecified) information and expertise” should get an extremely low grade (evaluation		

is also hard, because no difference is specified between different cases with no crashes). A very high grade means that all of these things need to be specified to a very high level of detail: i.e. a computer needs to know *all* decisions that need to be made and does not have a human's common sense, so nothing can be left implicit.		
Performance Evaluation	**Grade:**	**Conf:**
This will be 0 if performance evaluation is completely subjective; 10 if it can be captured by a simple algorithm. If this is high then training can be more easily automated; the lower, the more probable the need for a human trainer/teacher.		
Data availability	**Grade:**	**Conf:**
Important dimensions for data availability are 1) how much is available, 2) whether it contains all relevant variables, 3) the temporal nature of the data, 4) whether it is easy to work with / extract / adjust, 5) whether it is already available or can be obtained easily, 6) whether data is only from normal/random/boring scenarios or geared towards teaching. Simulators can help generate lots of new data and provide a great sandbox for learning control, especially if success can be quantified and evaluated in real time.		
Benefit from ML	**Grade:**	**Conf:**
This should be a 0 if it makes absolutely no sense to use machine learning for this task, a 10 when it absolutely requires ML, and something like a 5 if ML is not necessary, but it's not completely ridiculous to use it.		
Desirability for Isavia	**Grade:**	**Conf:**
This should be graded as "how much Isavia wants this task automated". This is purely *their* (fact-based) opinion, so we (RU) can only estimate it based on our talks. Here we should not take into account the value of succeeding at some abstract task and then later on being able to do something more directly useful, and also not the odds of success (that is already part of other criteria).		
Overall	**Grade:**	**Conf:**
Here an overall conclusion and possibly verdict should be written. The grade does not necessarily need to be an average of the previous ones.		

Scenarios

Here we should describe scenarios / use cases / (user) stories of what a typical sequence of events and decisions may look like under different circumstances. Such scenarios are used to determine what decisions are made, and should be annotated with the decision tags used in the next section.

Decisions

Here we should describe, in as much detail as possible, all decisions that are involved in carrying out the Task. For each decision, this involves answering:

- What is the **input**? What variables / information can or must be taken into account?

- What is the **output** or result? This can be anything, ranging from e.g. “a message to pilot X to move up/down by Y amount at time Z” to “preparation/prediction of the information for another decision”.
- By what **method** do we transform input to output? This can be a straightforward series of steps or calculations, or a vague description of an intuitive process.
- How can the decision be **evaluated**? What variables are being optimized? What is their relative importance?
- Optionally: what **features** does the decision have that could be used for decomposition?

Ideally, no decisions should be left implicit. There can be some redundancy due to describing decisions at both high and low levels (e.g. one decision may be “tell pilot what to do, based on all data”, which may involve decisions like “decide which pilot to talk to, based on closeness to airport”, “predict closeness to airport, based on weather”, etc.).

Decomposition

Here we should make a graphical representation of the task / decision hierarchy described in the previous section. This decision-based decomposition can (later) be annotated with feature-based and functionality-based decompositions. These decompositions should be described and motivated.

In the example diagram below we have a hypothetical task with five decisions. The decision-based decomposition shows that decision D1’s method directly involves the use of D2, D3 and D5, decision D2 uses D4, and decision D3 uses D4 and D5. The feature-based decomposition shows that decision D1 has 4 features (named a, b, c and d), decision D2 has three, and D3 has five, while no feature-based decomposition was performed for D4 and D5. The functionality-based decomposition shows that one “block” of functionality could consist of D2 (using only features a and b) and D4 (orange), and that another functionality block consists of D3, D4 and D5. More information on task decompositions is provided in the section ”Task Decomposition” above.

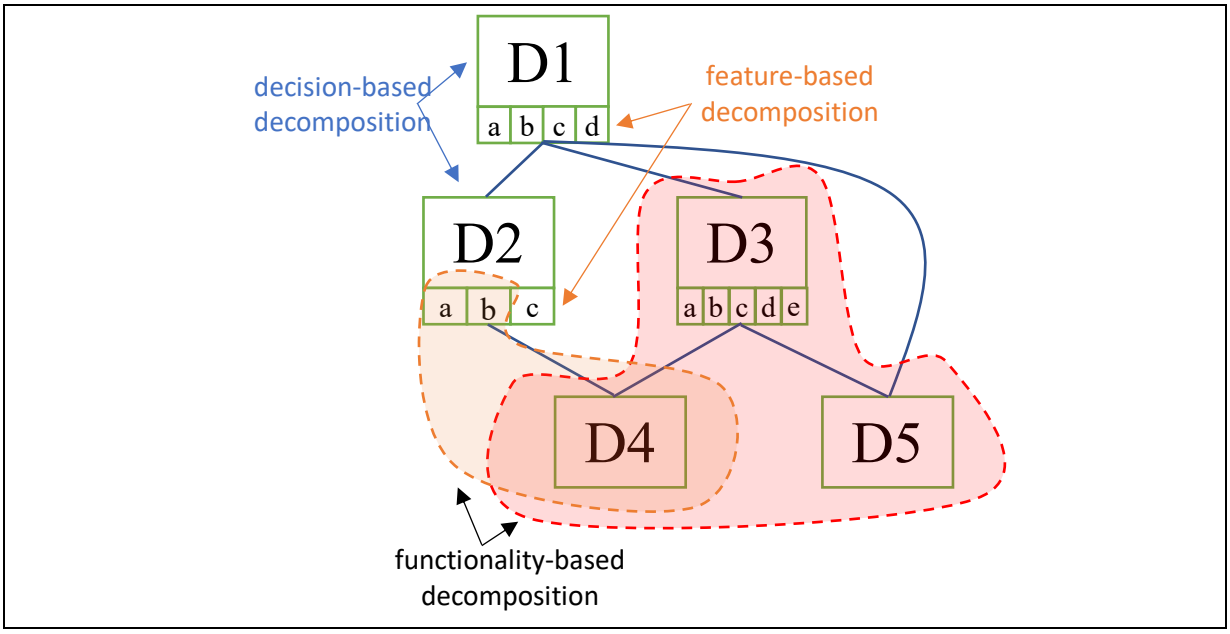


Figure showing various methods of task decomposition.

Task evaluation template

It starts with a level 2 heading with the title of the task ("Task evaluation template" here) followed by a **short description** of the task (this paragraph). Then we go into more depth by considering scenarios / (user) stories, detailed decisions and how they form a task hierarchy. Afterwards an evaluation follows by filling out the match rubric. However, it is typically wise to start filling it out with best guesses, and skipping the more detailed analysis in case the current HLT does not look promising or as promising as others.

Scenarios in Task

Scenario S1: Name

Description with references to decisions (D1) listed below (D2).

Scenario S2: Name

Description with references to (other) decisions (D1.1) listed below (D1).

Decisions in Task

Decision D1: Name

Short description...

Input:
Output:
Method:
Evaluation:

> *Decision D1.1: Name*
>
> Short description...
>
> **Input**:
> **Output**:
> **Method**:
> **Evaluation**:

Decision D2: Name

Short description...

Input:
Output:
Method:
Evaluation:

Decomposition of Task

Here the task decomposition hierarchy is described based on the analysis of decisions above.

Evaluation of Task

Achievability	**Grade:**	**Conf:**
Learnability	**Grade:**	**Conf:**
Teachability	**Grade:**	**Conf:**
Decomposability	**Grade:**	**Conf:**
Scalable complexity	**Grade:**	**Conf:**
Definability	**Grade:**	**Conf:**
Performance Evaluation	**Grade:**	**Conf:**
Data availability	**Grade:**	**Conf:**

Benefit from ML	**Grade:**	**Conf:**
Desirability for Isavia	**Grade:**	**Conf:**
Overall	**Grade:**	**Conf:**

Appendix B: Task Analyses

1. Candidate Task 1: Arrival Control

The primary goal of Arrival Control (AC / AMAN) is to ensure an optimal flow of aircraft arrivals at the airport. There must be a certain minimum amount of time separating two aircraft landing on the same runway, in addition to dynamic deviations due to things like other runway occupancy (e.g. for take-off). For a given runway, real-time predictions must be made about when each aircraft will arrive given the current 3D coordinates and speed, the flight plan, aircraft specs and weather conditions. If conflicts are detected or opportunities for improved arrival flow identified, the primary means of intervention is to ask a pilot to speed up or slow down. When a change to a flight occurs (or is requested), the consequences must be predicted, potential problems must be identified, a coping plan must be devised and a value judgement must be rendered (i.e. "no problem", "requires cheap intervention" or "requires very costly intervention").

Scenarios in Arrival Control

Scenario S1: Conflict resolution

The System is presented with information about all aircraft, the weather, etc. and needs to predict the time at which each aircraft is expected to arrive at each runway (D1). Based on this (and other) information, the System produces a list of likely conflicts (D2). The main job of the System is to resolve all conflicts (D3) by telling aircraft to speed up or slow down. Conflicts tend to be fixed one by one (D4). First a tentative decision must be made about which conflict to focus on (D5). Speed up / slow down commands for one aircraft or more must be selected (D6) and evaluated (D7). If a solution solves the conflict without introducing more, it must be verified with the main ATCO (D8) and the aircraft pilot (D8). If the candidate solution is accepted, it can be finalized (D9). If no solution can be found that doesn't introduce new conflicts, the System can try to solve these new conflicts as well in a similar manner (D6). At any point, the urgency of the currently evaluated conflict compared to others can be re-evaluated, and the System may switch to a different conflict (D5).

Scenario S2: Requests

A request comes in for a change to a certain aircraft's speed (D10). The System must calculate the new hypothetically expected arrival time (D1), evaluate its desirability (D7), and communicate it back to the requestor (D11).

Scenario S3: Monitoring

The System is constantly monitoring all aircraft in the airspace (D12). Changes are evaluated (D13) and incorporated in the System's current model (D14). Likely changes are predictable updates to aircraft's location/speed, but they can also be more unexpected like a new aircraft entering the airspace or an error about some aircraft being fixed. If a change was unexpected (D15), new arrival times will need to be calculated (D1), conflicts detected (D2) and resolved (D3).

Decisions in Arrival Control

Decision D1: Arrival time prediction

Predict the time at which aircraft A will arrive at runway R.

Input: aircraft info for A (including coordinates (X, Y, Z) and speed (dX, dY, dZ), the flight plan, aircraft specs), runway R, and weather
Output: time (+ maybe confidence, or a distribution over times)
Method: Maybe look at flight plan and adjust for currently known delays + heuristics about weather
Evaluation: (predicted time - actual time sans interventions)^2

Decision D2: Conflict detection

Detect possible conflicts in arrival times for runway R.

Input: minimum time window, runway R, and a sorted list of estimated arrival times (+ maybe confidences, or a list of time distributions)
Output: list of conflicting pairs ordered by likelihood of conflict and urgency
Method: Go over list and see where the separation is too low
Evaluation: accuracy

Decision D3: Global conflict resolution

Resolve conflicts between aircraft arrival times by issuing commands to some of them.

Input: aircraft info, predicted arrival times, weather and conflicts
Output: list of speed up / slow down commands for various aircraft
Method: Select a conflict based on priority and resolve it
Evaluation: based on number of resolved (and introduced) conflicts and cost of interventions (which includes their number, magnitude, etc.)

Decision D4: Single conflict resolution

Given a conflict, resolve it.

Input: same as D3 + conflict to focus on
Output: list of speed up / slow down commands for aircraft (possibly multiple, but likely just one)
Method: assess whether it's better to speed up the first aircraft or slow down the last one
Evaluation: based on cost of interventions (which includes their number, magnitude, etc.)

Decision D5: Conflict urgency evaluation

Evaluate the urgency of a given conflict.

Input: same as D3 + conflict to focus on
Output: urgency value
Method: take the closest conflicting pair
Evaluation: indirect by evaluating reward received for parent task

Decision D6: Resolution candidate generation

Generate a promising candidate solution for solving the given conflict.

Input: same as D3 + conflict to focus on + history of tried hypotheses
Output: speed up / slow down command for aircraft (possibly multiple, but likely just one)
Method: intuition / try random values of speed up or slow down
Evaluation: originality or proposal / proposal acceptance rate and speed / optimality of proposals / calculate how much the global conflict badness score is affected

Decision D7: Resolution candidate evaluation

Generate a promising candidate solution for solving the given conflict.

Input: same as D3 + conflict to focus on + candidate solution from D6
Output: whether it solves the conflict and introduces different ones
Method: calculate how much the global conflict badness score is affected
Evaluation: accuracy

Decision D8: Candidate resolution proposal

Propose the candidate solution to others (ATCO & pilot), and wait for response.

Input: aircraft + candidate command
Output: whether the proposal was approved
Method: use comms
Evaluation: delivery rate / user feedback

Decision D8: Request change evaluation from aircraft

Request a desirability value for certain change from an aircraft (e.g. speed up or slow down).

Input: aircraft, change
Output: message to aircraft
Method: use comms
Evaluation: delivery rate / quality of answers / user feedback

Decision D9: Candidate resolution implementation

Command all aircraft to implement the candidate solution and notify others (e.g. ATCO)

Input: aircraft + candidate command
Output: whether the proposal was approved
Method: use comms
Evaluation: delivery rate / user feedback / compliance

Decision D10: Request change from aircraft

Request a certain change from an aircraft (e.g. speed up or slow down).

Input: aircraft, change
Output: message to aircraft
Method: use comms
Evaluation: delivery rate / user feedback

Decision D11: Request handling

A request for a change for some aircraft comes in (e.g. to speed / route) and must be handled.

Input: Same as D6 + request value
Output: approval / dismissal + reason
Method: Approve if desirable according to D5, otherwise compare undesirability to “request value”. Dismiss if not urgent. Otherwise, attempt conflict resolution with the hypothetical change, and compare request value to cost of approval.
Evaluation: see D8 and D3

Decision D12: Change handling

Respond to a change in some information about an aircraft.

Input: Same as D6 + change for one aircraft (speed, location, type, course, etc.)
Output: incorporate change into knowledge (D14) and resolve new conflicts (D1,D2,D3)

Method: use D13, D14 and D15
Evaluation: use overall performance rewards

Decision D13: Change surprise evaluation
Evaluates whether a change is surprising or in line with predictions.

Input: Same as D12
Output: surprise value
Method: calculate dissimilarity between prediction and observation
Evaluation: accuracy of dissimilarity

Decision D14: Change incorporating
Incorporate a change into the system's current knowledge.

Input: Same as D12
Output: updated arrival time predictions, conflict detections
Method: depends on AI system
Evaluation: accuracy

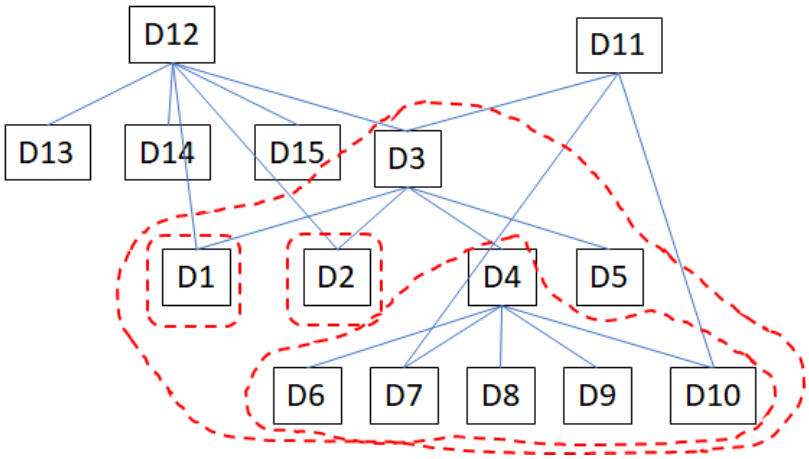
Decision D15: Unexpected change handling
Respond to a surprising change in some information about an aircraft.

Input: Same as D12
Output: incorporate change into knowledge (D14) and resolve new conflicts (D1,D2,D3)
Method: see D14, D1, D2 and D3
Evaluation: see D14, D1, D2 and D3

Decision D16: Aircraft freedom range estimation
Estimate the range of freedom that a particular aircraft has to deviate from its current speed.

Input: same as D3 + aircraft to focus on
Output: range of acceptable speeds that would still avoid conflict
Method: find two closest aircraft and perform basic arithmetic
Evaluation: accuracy

Decomposition of Arrival Control



Example Evaluation of Arrival Control Using Proposed Evaluation Methodology

Achievability	**Grade:** 8	**Conf:** 5
The relatively low dimensionality of simple versions of this task should make it doable to use this Task as a prototype for the methodology we aim to develop.		
Learnability	**Grade:** 6	**Conf:** 5
We predict that the task can be learned, but that the prediction part may be rather monolithic and the interdependencies between a variable amount of aircraft will add significant complexity.		
Teachability	**Grade:** 7	**Conf:** 2
This mostly remains to be seen. Intermediate rewards could likely be given based on individual conflict resolutions. Decomposition seems limited. We should be able to teach both with rewards and examples. It's not clear how much (de)simplification will help. It seems fairly easy to create "interesting" training scenarios.		
Decomposability	**Grade:** 5	**Conf:** 5
The Task seems rather monolithic to us. The main tasks are arrival time prediction and conflict resolution. Both seem difficult to divide into many learnable subtasks, although there are some possibilities with conflict resolution (urgency identification, hypothesis generation, evaluation, etc.). Decomposition into e.g. sectors and kinds of weather should also be possible, but doesn't help much with hierarchical reinforcement learning.		
Scalable complexity	**Grade:** 9	**Conf:** 7
The simplest version of this task is rather a rather simple 1D task (e.g. given arrival predictions, tell each airplane to slow down or speed up), but it can be scaled up to include e.g. weather, higher dimensions, noise, difficult pilots/ATCOs, complex costs, etc.		
Definability	**Grade:**	**Conf:**

	9	8
It is pretty clear what the task is, what data is needed and what actions need to be taken.		
Performance Evaluation	**Grade:** 8	**Conf:** 7
It seems to us that an algorithm could be created that uses e.g. average distance between airplanes and whether all of them are larger than the minimum time window. Little effort (fewer commands) is better than many, for any time window. Maybe qualified by the difficulty (number of airplanes, windspeed).		
Data availability	**Grade:** 9	**Conf:** 2
Our understanding is that a good amount of data is available. Isavia showed us they keep a database of related information. However, we need to dig deeper to understand what it is exactly, and how good/useful it is.		
Benefit from ML	**Grade:** 6	**Conf:** 6
Some judgment is involved in especially the prediction part, but the planning part can probably be accomplished fairly well with existing planning algorithms.		
Desirability for Isavia	**Grade:** 8	**Conf:** 2
Both Isavia and Tern came to us with this Task and we got the impression they would find automation useful.		
Overall	**Grade:** 8.5	**Conf:** 5
Overall this Task seems fairly suitable for our artificial pedagogical approach.		

2. Candidate Task 2: Conflict Resolution

The primary goal of Conflict Resolution (CR) is to ensure that minimum separation rules between aircraft in the airspace are maintained so as to avoid collisions. This requires real-time predictions of the trajectories of all aircraft given their current 3D coordinates and speed, the flight plan, aircraft specs and weather conditions. If a potential conflict between aircraft is detected, the ATCO can request one or both to slightly change course (mostly left/right, seldomly up/down), which would presumably replace some "checkpoint" in the flight plan. The effects for (requests for) changes to flight plans must be predicted and if they introduce a conflict, they can be rejected (possibly after estimating the difficulty of resolving the conflict).

Scenarios in Conflict Resolution

Scenario S1: Separation maintenance

The system is presented with information about all aircraft, the weather, etc. This information must be incorporated in memory in order to make predictions about each aircraft's trajectory while it is in the airspace (D1), at least N minutes into the future, but ideally until it is predicted to leave the airspace. The System needs to decide in what order to update the trajectories for each aircraft (D1.1; I'm using weird numbering to preserve similarity with AC). Based on these predictions, the System must produce a list of likely conflicts (D2), after which its main task is to resolve them (D3). This tends to happen one by one (D4) after priorities have been assigned to each conflict (D5). Functionality may exist that first estimates an area for attention (D2.1).

Conflicts are resolved by issuing commands to aircraft to (usually) divert to the left or right or (sometimes) up or down. Candidate solutions must be generated (D6) and evaluated (D7). If the candidate seems promising, it must be communicated to and approved by the involved aircraft and potentially other ATCOs (D8), after which the final command can be issued (D9). The urgency of any conflict can be re-evaluated at any time, at which point the System may switch to resolving a different one (D5).

Scenario S2: Requests

A request comes in for a change to a certain aircraft's speed or course. The System must calculate the new expected trajectory (D1), evaluate its desirability (D7), and communicate it back to the requestor (D11).

Scenario S3: Monitoring

The System is constantly monitoring all aircraft in the airspace (D12). Changes are evaluated (D13) and incorporated in the System's current model (D14). Likely changes are predictable updates to aircraft's location/speed, but they can also be more unexpected like a new aircraft entering the airspace or an error about some aircraft being fixed. If a change was unexpected (D15), new arrival times will need to be calculated (D1), conflicts detected (D2) and resolved (D3).

Decisions in Conflict Resolution

Question marks indicate missing data. Text following question marks represent a proposal for that item.

Decision D1: Trajectory prediction

Predict the trajectory of aircraft A for the foreseeable future.

Input: aircraft info for A (including coordinates (X, Y, Z) and speed (dX, dY, dZ), the flight plan, aircraft specs), runway R, and weather
Output: trajectory (where the aircraft will be at what time)
Method: TBD – Maybe look at flight plan and adjust for currently known delays + heuristics about weather
Evaluation: accuracy compared to actually flown route

Decision D1.1: Trajectory prediction priority assessment

Decide for which aircraft the trajectory should be predicted / updated next.

Input: aircraft info for all (including coordinates (X, Y, Z) and speed (dX, dY, dZ), flight plans, aircraft specs), current predictions, and weather
Output: an aircraft ID
Method: Possibly look at which predictions are oldest, which aircraft are close to others, and which are close to landing
Evaluation: Whether the System succeeds in predicting trajectories before it is too late

Decision D2: Conflict detection

Detect possible separation conflicts.

Input: minimum separation window, and a list of trajectories + D1's input
Output: list of conflicting pairs ordered by likelihood of conflict and urgency
Method: Order aircraft by proximity and check for intersections in trajectory
Evaluation: accuracy

Decision D2: Area of attention detection

Mark an area that should be paid attention to.

Input: minimum separation window, and a list of trajectories + D1's input
Output: an area where conflicts will be more likely to occur
Method: Density estimation or possibly something else
Evaluation: <undefined>

Decision D3: Global conflict resolution

Resolve conflicts between aircraft arrival times by issuing commands to some of them.

Input: aircraft info, predicted arrival times, weather and conflicts
Output: list of commands for various aircraft
Method: TBD
Evaluation: based on number of resolved (and introduced) conflicts and cost of interventions (which includes their number, magnitude, etc.)

Decision D4: Single conflict resolution

Given a conflict, resolve it.

Input: same as D3 + conflict to focus on

Output: list of commands for aircraft (possibly multiple, but likely just one)
Method: TBD
Evaluation: based on cost of interventions (which includes their number, magnitude, etc.)

Decision D5: Conflict urgency evaluation

Evaluate the urgency of a given conflict.

Input: same as D3 + conflict to focus on
Output: urgency value
Method: TBD
Evaluation: TBD

Decision D6: Resolution candidate generation

Generate a promising candidate solution for solving the given conflict.

Input: same as D3 + conflict to focus on + history of tried hypotheses
Output: command for aircraft (possibly multiple, but likely just one)
Method: TBD
Evaluation: TBD

Decision D7: Resolution candidate evaluation

Generate a promising candidate solution for solving the given conflict.

Input: same as D3 + conflict to focus on + candidate solution from D6
Output: whether it solves the conflict and introduces different ones
Method: TBD
Evaluation: TBD

Decision D8: Candidate resolution proposal

Propose the candidate solution to others (ATCO & pilot), and wait for response.

Input: aircraft + candidate command
Output: whether the proposal was approved
Method: TBD
Evaluation: TBD

Decision D8: Request change evaluation from aircraft

Request a desirability value for certain change from an aircraft (e.g. speed up or slow down).

Input: aircraft, change
Output: message to aircraft
Method: TBD
Evaluation: TBD

Decision D9: Candidate resolution implementation

Command all aircraft to implement the candidate solution and notify others (e.g. ATCO)

Input: aircraft + candidate command
Output: whether the proposal was approved
Method: TBD
Evaluation: TBD

Decision D10: Request change from aircraft

Request a certain change from an aircraft (e.g. speed up or slow down).

Input: aircraft, change
Output: message to aircraft
Method: TBD
Evaluation: TBD

Decision D11: Request handling

A request for a change for some aircraft comes in (e.g. to speed / route) and must be handled.

Input: Same as D6 + request value
Output: approval / dismissal + reason
Method: Approve if desirable according to D5, otherwise compare undesirability to "request value". Dismiss if not urgent. Otherwise, attempt conflict resolution with the hypothetical change, and compare request value to cost of approval.
Evaluation: see D8 and D3

Decision D12: Change handling

Respond to a change in some information about an aircraft.

Input: Same as D6 + change for one aircraft (speed, location, type, course, etc.)
Output: incorporate change into knowledge (D14) and resolve new conflicts (D1,D2,D3)
Method: use D13, D14 and D15
Evaluation: TBD

Decision D13: Change surprise evaluation

Evaluates whether a change is surprising or in line with predictions.

Input: Same as D12
Output: surprise value
Method: TBD
Evaluation: TBD

Decision D14: Change incorporating

Incorporate a change into the system's current knowledge.

Input: Same as D12
Output: updated arrival time predictions, conflict detections
Method: TBD
Evaluation: accuracy

Decision D15: Unexpected change handling

Respond to a surprising change in some information about an aircraft.

Input: Same as D12
Output: incorporate change into knowledge (D14) and resolve new conflicts (D1,D2,D3)
Method: TBD
Evaluation: TBD

Decision D16: Aircraft advice

For a particular aircraft, issue advice for what to do to the ATCO

Input: same as D3 + aircraft to focus on
Output: a command + urgency value
Method: TBD
Evaluation: accuracy

Decomposition of Conflict Resolution

TBD

Evaluation of Conflict Resolution

Achievability	**Grade:** 5	**Conf:** 5
The higher dimensionality make the simplest version of this task more difficult than AC, and it may be tricky to get something resembling real conflict resolution to work within the time period.		
Learnability	**Grade:** 8	**Conf:** 5
We think this can be learned. Certainly in its simplest form.		
Teachability	**Grade:** 7	**Conf:** 2
This mostly remains to be seen. Intermediate rewards could likely be given based on individual conflict resolutions. Decomposition seems possible. We should be able to teach both with rewards and examples. It's not clear how much (de)simplification will help. It seems fairly doable to create "interesting" training scenarios.		
Decomposability	**Grade:** 7	**Conf:** 5
We think the task can be decomposed into steps like "identify potential problem area", "predict aircraft trajectory", "detect pairwise conflicts", "resolve conflict", ...		
Scalable complexity	**Grade:** 8	**Conf:** 7
The simplest version would be to simply consider aircraft within a certain distance of each other pairwise. Complexity can be scaled up from here, but it is slightly more complicated than in AC, because this is at least 2D.		
Definability	**Grade:** 7	**Conf:** 8

Overall it is pretty clear, but (higher dimensional) trajectories and changes to them are harder to evaluate than 1D speeds.		
Performance Evaluation	**Grade:** 7	**Conf:** 7
Probably a little more difficult than AC, but not overly so.		
Data availability	**Grade:** 7	**Conf:** 2
Our understanding is that a good amount of data is available. However, it is not clear whether all variables that are required are recorded. Since this Task is less structured, it is more difficult to know if this data is good or sufficient.		
Benefit from ML	**Grade:** 9	**Conf:** 6
Possibly a bit more complex than AC, but probably not much.		
Desirability for Isavia	**Grade:** 7	**Conf:** 2
Both Isavia and Tern came to us with this Task and we got the impression they would find automation useful, but they were more excited about AC.		
Overall	**Grade:** 8	**Conf:** 5
Overall this Task seems fairly suitable for our artificial pedagogical approach, but harder to implement in our timeline.		

3. Candidate Task 3: Workload Optimization (stress/boredom management)

The main goal of the Workload Optimization (WO) Task is to estimate the amount of work that would be neither too boring nor too stressful for each individual ATCO, and to then help to achieve this optimal workload. The inverted U theory states that performance is low when arousal is either too low or too high, and optimal somewhere in the middle. The exact shape of the arousal-performance curve is different for each person, and may even differ per day. Essentially, we want to monitor (and possibly control) a person's work pressure so that it does not exceed a point where performance will drop drastically. When this happens, a warning could be sent to the ATCO and/or staff manager, or the workload can be lessened by dividing the airspace into smaller pieces (or expanding it, if the ATCO is bored) or taking over part of the work with automation. The task involves "estimating work pressure", "predicting performance, given pressure", and "predicting how pressure is affected by several coping methods". Pressure estimates could be learned based on (statistics about) the inputs an ATCO receives, or based on physiological and behavioral measurements. We would need (probably a lot) of data that includes performance evaluations. This task also seems difficult to simplify and (further) decompose, so I'm not sure it is a good idea at this point in time.

Scenarios in Workload Optimization

Scenario S1: Workload monitoring

We estimate an ATCO's workload (D1) and (possibly) physiological state (D2) and performance (D3). This is then combined to estimate the ATCO's mental state (D4) and compared to that ATCO's desired level (D5). This information can then be displayed for the ATCO (D6) so they can act on it themselves or communicated to a manager (D7).

Scenario S2: Workload balancing

Based on the ATCO's mental state, we could also decide that workload rebalancing is in order. Workload decrease could e.g. be done by taking over part of the task (D8). Workload increase could be done by stopping to take over certain parts (D9), adding arbitrary distractors (D10), or expanding the area of control (D11).

Scenario S3: Personalization

We need to figure out each ATCO's individual inverted U curve.

...

Decisions in Workload Optimization

Decision D1: Workload estimation

Estimate the workload of an ATCO.

Input: aircraft info, weather info
Output: scalar value estimating absolute workload
Method: TBD
Evaluation: TBD

Decision D2: Physiological state estimation

Estimate the physiological state of an ATCO.

Input: GSR / computer vision / EEG?
Output: scalar value estimating physiological state of boredom/stress/confusion
Method: GSR / computer vision / EEG / TBD

Evaluation: TBD

Decision D3: Performance estimation

Estimate the current performance of an ATCO.

Input: Possibly performance measure (as used in CR / AC), e.g. number of aircraft that violated separation / TBD
Output: scalar value estimating current performance
Method: TBD
Evaluation: TBD

Decision D4: Mental state estimation

Estimate the mental state of an ATCO in terms of boredom/stress/confusion.

Input: output of D1, D2 and D3
Output: scalar value estimating mental state of boredom/stress/confusion
Method: TBD
Evaluation: TBD

Decision D5: Mental state evaluation

Evaluate the position of mental state of an ATCO's mental state compared to the ideal one on the inverted U curve for that ATCO.

Input: output of D4, ATCO's inverted U curve
Output: value indicating whether the workload is too low/high
Method: TBD
Evaluation: TBD

Decision D6: Displaying current ATCO state

Outputs of D1-5 could be displayed in the ATCO's interface.

Input: output of D1-5
Output: TBD
Method: TBD
Evaluation: TBD

Decision D7: Manager warning

Outputs of D1-5 could be communicated to the ATCO's manager.

Input: output of D1-5
Output: message to manager
Method: TBD
Evaluation: TBD

Decision D8: Workload takeover

Taking over part of the ATCO's workload

Input: TBD (everything the ATCO sees)
Output: CR/AC, removal of task parts for ATCO
Method: CR/AC
Evaluation: TBD

Decision D9: Workload relinquishment

Relinquish part of the workload that had been taken over back to the ATCO.

Input: TBD (everything the ATCO sees)
Output: addition of task parts for ATCO
Method: TBD
Evaluation: TBD

Decision D10: Workload increase through arbitrary distractors

Add some kind of distractor / challenge to prevent boredom.

Input: TBD
Output: TBD (distractors...)
Method: TBD
Evaluation: TBD

Decision D11: Workload increase through task expansion

Expand the ATCO's area of control and/or tasks.

Input: TBD
Output: TBD (A plan for how to do this, probably communicated to manager)
Method: TBD
Evaluation: TBD

Decomposition of Workload Optimization

TBD

Evaluation of Workload Optimization

Achievability	**Grade:** 3	**Conf:** 5
This project seems difficult to accomplish with just one AI researcher within the desired time frame, since it likely requires large amounts of data collection which may even be physiological and psychological in nature.		
Learnability	**Grade:** 7	**Conf:** 5
It seems like with the right data, this should be somewhat learnable. However, that data availability will be a problem.		
Teachability	**Grade:** 3	**Conf:** 5
This seems difficult, because it's not really clear how well the task can be decomposed or simplified, and it will be difficult to give intermediate rewards (for what?).		

Decomposability	**Grade:** 4	**Conf:** 5
We can decompose the task into “estimating work pressure”, “predicting performance, given pressure”, and “predicting how pressure is affected by several coping methods”, but each of those seem rather monolithic.		
Scalable complexity	**Grade:** 2	**Conf:** 7
This task is very hard to simplify, because we can’t simplify the human element (which is very important for this task). We cannot consider less tasks for instance, or much simpler ones, because then we would automatically be on everybody’s left side of the inverted U.		
Definability	**Grade:** 7	**Conf:** 4
The task itself seems reasonably well defined, but in practice each instance of the task will depend heavily on hard to measure human factors.		
Performance Evaluation	**Grade:** 5	**Conf:** 7
ATCO performance evaluation seems tricky but possible. Even more trickier would be to evaluate subtasks.		
Data availability	**Grade:** 3	**Conf:** 5
Estimations could either be made based on (statistics about) the inputs an ATCO receives, or based on physiological and behavioral measurements. Data for that latter option is almost certainly not available, and difficult to get. We may have input data for ATCOs, but it’s not clear that we also have an idea of how well they performed on it.		
Benefit from ML	**Grade:** 9	**Conf:** 6
Probably impossible to do without ML.		
Desirability for Isavia	**Grade:** 6	**Conf:** 2

Our impression is that this would be nice to have, but mostly a curiosity. It is not part of the essential workflow.		
Overall	**Grade:** 4	**Conf:** 5
Overall this Task seems unsuitable for our approach.		

4. Candidate Task: Directive Adaptation Assistance

The Directive Adaptation Assistance (DAA) Task would help ATCOs take all (new) directives into account in their work. The System might get some kind of description of the directive (likely in some machine readable format, but eventually perhaps in natural language), and help in recognizing situations where it applies or is violated, at which point the ATCO could be notified. This may require that the AI already "understands" the main task, so we may need to leave this until later. It is worth noting that most current AI systems would need to be retrained (likely from scratch) to take a change in directives / task parameters into account, whereas AERA and NARS should be able to incorporate (and later forget) them relatively easily.

Scenarios in Directive Adaptation Assistance

Scenario S1: Separation Constraints

A new directive is added that changes separation constraints. Somehow, this directive is fed into the System. The System then interprets the directive (D1), and relays it to the ATCO through some kind of message (D2). The System also tries to keep track of what the ATCO does (D3), and predicts when actions are not in line with the directive (D4), so that additional warnings can be shown (D5).

Scenario S2: another directive...

TBD

Decisions in Directive Adaptation Assistanceh

TBD

Decomposition of Directive Adaptation Assistance

TBD

Evaluation of Directive Adaptation Assistance

TBD

Achievability	**Grade:** 1	**Conf:** 5
Learning to deal with all possible directives would require a near-general AI, which cannot be developed within the envisioned time frame. (More limited systems may be doable, but may not need machine learning.)		
Learnability	**Grade:** 2	**Conf:** 5
If we accept all possible directives, that's basically a Turing complete language, and we would need to		

have the right response for everything, which is virtually impossible.		
Teachability	**Grade:** 2	**Conf:** 5
Each directive would pose a new “task”, which would need to be taught automatically, and we don’t know how to do that.		
Decomposability	**Grade:** 2	**Conf:** 3
Each directive would pose a new “task”, which would need to be decomposed automatically, and we don’t know how to do that.		
Scalable complexity	**Grade:** 4	**Conf:** 5
I suppose it’s possible to work with only certain kinds of directives, but it’s not clear how this scales.		
Definability	**Grade:** 2	**Conf:** 5
Each directive would essentially pose a new “task”, which makes this task a bit ill-defined.		
Performance Evaluation	**Grade:** 4	**Conf:** 6
Each directive might have a different (doable) evaluation method attached. As a whole, the Task would be almost impossible to evaluate.		
Data availability	**Grade:** 4	**Conf:** 2
Directives are likely available, but probably not in machine-readable form.		
Benefit from ML	**Grade:** 5	**Conf:** 3
Depends on the directive, but the Task as a whole is probably very difficult to learn.		

Desirability for Isavia	**Grade:** 6	**Conf:** 2
Isavia briefly mentioned this as a possible Task.		
Overall	**Grade:**2	**Conf:** 5
Overall this Task does not seem suitable for our approach.		

BIBLIOGRAPHY

A. Barry. A hierarchical XCS for long path environments. In Proceedings of the Third Genetic and Evolutionary Computation Conference (GECCO-2001), pages 913–920, 2001.

J. Bieger, K. R. Thórisson, and D. Garrett. Raising AI: Tutoring Matters. In Proceedings of AGI-14, pages 1–10, Quebec City, Canada, 2014.

J. Bieger, K. R. Thórisson, and B. R. Steunebrink. Evaluation of General-Purpose Artificial Intelligence: Why, What & How. In Evaluating General-Purpose AI 2016, The Hague, Netherlands, 2016.

J. Bieger, K. R. Thórisson, and B. R. Steunebrink. Evaluating understanding. In IJCAI Workshop on Evaluating General-Purpose AI, Melbourne, Australia, 2017.

P. Flener and U. Schmid. An introduction to inductive programming. Artificial Intelligence Review, 29(1):45–62, 2008.

I. Goodfellow, Y. Bengio, and A. Courville. Deep learning. MIT press, 2016.

B. Hengst. Hierarchical approaches. In Reinforcement learning, pages 293–323. Springer, 2012.

S. B. Kotsiantis. Decision trees: a recent overview. Artificial Intelligence Review, 39(4):261–283, 2013.

E. Nivel, K. R. Thóorisson, B. R. Steunebrink, H. Dindo, G. Pezzulo, M. Rodriguez, C. Hernandez, D. Ognibene, J. Schmidhuber, R. Sanz, H. P. Helgason, A. Chella, and G. K. Jonsson. Bounded Recursive Self-Improvement. Technical RUTR-SCS13006, Reykjavik University Department of Computer Science, Reykjavik, Iceland, 2013.

Richard S. Sutton and Andrew G. Barto. Introduction to reinforcement learning. MIT Press, 1998.

K. R. Thórisson, J. Bieger, S. Schiffel, and D. Garrett. To wards a Task Environment Description Language for Comprehensive Evaluation of Artificial Intelligent Systems & Automatic Learners. Technical RUTR-SCS15001, Reykjavik University School of Computer Science, Reykjavik, Iceland, 2015.

P. Wang. Non-Axiomatic Logic: A Model of Intelligent Reasoning. World Scientific Publishing, Singapore, 2013.

S. W. Wilson. Classifier Fitness Based on Accuracy. Evolutionary Computation, 3(2):149–175, 1995.