



AFRL-RH-WP-TR-2019-0093

**REVIEW OF AUTONOMY/AGENT
SYSTEM
ALGORITHMS AND TECHNIQUES TO
INFORM OPERATOR INTERFACE DESIGN**

**Jeffrey Duffy
Thomas C. Hughes, Ph.D.**

**Infoscitex Corp.
4027 Colonel Glenn Highway, Suite 210
Dayton, OH 45431**

August 2019

Final Technical Report

Distribution A: Approved for public release: distribution is unlimited.

**AIR FORCE RESEARCH LABORATORY
711 HUMAN PERFORMANCE WING
AIRMAN SYSTEMS DIRECTORATE
WARFIGHTER INTERFACE DIVISION
WRIGHT-PATTERSON AIR FORCE BASE, OH 45433
AIR FORCE MATERIEL COMMAND
UNITED STATES AIR FORCE**

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

Qualified requestors may obtain copies of this report from the Defense Technical Information Center (DTIC).

AFRL-RH-WP-TR-2019-0093 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

//signature//

GLORIA L. CALHOUN, M.A., DR-IV
Principal Engineering Research Psychologist
Collaborative Interfaces & Teaming Branch
Airman Systems Directorate
711th Human Performance Wing
Air Force Research Laboratory

//signature//

TIMOTHY WEBB, Ph.D., CHIEF
Collaborative Interfaces & Teaming Branch
Warfighter Interface Division
Airman Systems Directorate
711th Human Performance Wing
Air Force Research Laboratory

//signature//

LOUISE A. CARTER, Ph.D., DR-IV
Chief, Warfighter Interface Division
Airman Systems Directorate
711 Human Performance Wing
Air Force Research Laboratory

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

THIS PAGE IS INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE				Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing this collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.					
1. REPORT DATE (DD-MM-YYYY) 01/08/2019		2. REPORT TYPE Final		3. DATES COVERED (From - To) July 2014 – June 2019	
4. TITLE AND SUBTITLE Review of Autonomy/Agent System Algorithms and Techniques to Inform Operator Interface Design				5a. CONTRACT NUMBER FA8650-14-D-6500	
				5b. GRANT NUMBER	
				5c. PROGRAM ELEMENT NUMBER 62202F	
6. AUTHOR(S) Jeffrey Duffy, Thomas Hughes				5d. PROJECT NUMBER	
				5e. TASK NUMBER Task Order 002	
				5f. WORK UNIT NUMBER H0JC (53290904)	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Infoscitex Corporation, a DCS Company 4027 Colonel Glenn Highway, Suite 210 Dayton, OH 45431				8. PERFORMING ORGANIZATION REPORT NUMBER N/A	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Force Research Laboratory 711 th Human Performance Wing Airman Systems Directorate Warfighter Interface Division Collaborative Interfaces and Teaming Branch Wright-Patterson AFB, OH 45433				10. SPONSOR/MONITOR'S ACRONYM(S) 711 HPW/RHCC	
				11. SPONSOR/MONITOR'S REPORT NUMBER AFRL-RH-WP-TR-2019-0093	
12. DISTRIBUTION / AVAILABILITY STATEMENT Distribution A: Approved for public release: Distribution is unlimited.					
13. SUPPLEMENTARY NOTES This is the Final Technical Report for the Course of Action Tools for Decision Superiority (COATDS) effort. 88ABW-2020-0156; Cleared 22 Jan 2020					
14. ABSTRACT While the introduction of automated systems in complex work environments has had a significant impact on increasing the productivity and efficiency across many work domains, these impacts were not achieved without unintended consequences. The automation landscape is littered with countless stories of automation mishaps, many attributed to ineffective interfaces between automated systems and the operators that control them. Advances in artificial intelligence and the development and incorporation of autonomous systems in work environments pose similar challenges, with potentially significant consequences. This literature review and analysis attempts to gain a fundamental understanding of the tools and techniques used to develop autonomous systems as a precursor to the creation of interfaces that can support effective human-autonomy coordination and interaction. Understanding these various techniques provides insight into the information these systems can provide for better observability into optimization and the decision process as well as the extent to which operators are able to direct these processes in response to situational dynamics and uncertainty. A taxonomy was also developed to serve as a common language to support dialog between agent and interface developers regarding an agent's agility, directability, observability, and transparency (ADOT).					
15. SUBJECT TERMS Autonomy, Human-Machine Interface, Human-Autonomy, Human-Agent					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Ms. Gloria Calhoun
Unclassified	Unclassified	Unclassified	SAR	115	19b. TELEPHONE NUMBER (include area code)

Table of Contents

List of Figures	ii
List of Tables.....	ii
1.0 INTRODUCTION	1
2.0 ARTIFICIAL INTELLIGENCE/AUTONOMOUS SYSTEMS LITERATURE REVIEW..	3
2.1 Findings	7
3.0 TUTORIAL SESSIONS	14
4.0 AGENT DESCRIPTORS	15
4.1 Descriptor Definitions	16
4.1.1 Agility Definition.....	16
4.1.2 Directability Definition.....	17
4.1.3 Observability Definition.	18
4.1.4 Transparency Definition.	18
4.2 Descriptor Levels.....	19
4.2.1 Agility Levels.....	19
4.2.2 Directability Levels.....	21
4.2.3 Observability Levels.	23
4.2.4 Transparency Levels.	24
4.3 Taxonomy Summary.....	25
4.4 Multi-Agent System Analysis.....	28
4.4.1 Operator-Unmanned Vehicle Control System.	28
4.4.2 Sense and Avoid Agent (SAA).	33
4.4.3 Task Management Agent.	34
5.0 SUMMARY.....	37
6.0 REFERENCES	38
Appendix A - Bibliography	41
Appendix B - Tutorials	55
LIST OF ABBREVIATIONS AND ACRONYMS	111

List of Figures

Figure 1	Screenshot of a Portion of the Exported Spreadsheet of Tagged Documents.....	5
Figure 2	Screenshot of the Type Key and Beginning of the Exported Spreadsheet.....	6
Figure 3	Agent Capabilities Vs. Agent Requirements/Intended Use and Implementation	13
Figure 4	Sample Ground Route Shown On a Map in Simulation	29
Figure 5	Path Planning Agent Implementations compared to Taxonomic Levels of the Fundamental Dijkstra's Algorithm Assessment	31
Figure 6	Hierarchical Architecture of Supervisory Control System.....	32
Figure 7	Technique Ratings by Function.....	33
Figure 8	Task Manager for Operator and Agent Teammate Coordination.....	33

List of Tables

Table 1.	Literature Type and Number Gathered and Reviewed.....	4
Table 2	Tags Used During The Literature Review	10
Table 3.	Assessment of Techniques in Terms of Support for System <i>Agility</i>	26
Table.4.	Assessment of Techniques in Terms of Support for System <i>Directability</i>	26
Table 5.	Assessment of Techniques in Terms of Support for System <i>Observability</i>	27
Table 6.	Assessment of Techniques in Terms of Support for System <i>Transparency</i>	27

1.0 INTRODUCTION

Several factors are converging that are fundamentally reshaping how military weapon systems are conceived, developed, deployed and operated. First, the range of operational missions that our military forces support cover a broad spectrum of operational demands under an increasingly difficult range of operational environments. Second, the nature and pervasiveness of our adversaries, to include not only sovereign states but also non-state sanctioned terrorist organizations, threaten the stability of critical regions of interest to the United States (US) and the security of US interests both at home and abroad. Third, the scope and pace of current and future operations challenge our ability to effectively adapt and respond to situations in a timely manner. The ability to anticipate the actions of our adversaries and respond in a manner that negates or minimizes the impact of these actions will remain a primary objective of developing military capabilities. Finally, technology advances not only enhance our own capabilities to support military operations but also the technological sophistication of our enemies. Their ability to leverage the increasing availability of emerging technological capabilities in their own operations or, possibly more damaging, to exploit the vulnerabilities of our own technologies against us increases the emerging threat.

While the US continues to enjoy a significant technological advantage against those that threaten the US and our allies, that advantage is perpetually at risk. As the Department of Defense (DoD) attempts to deal with the increased complexity in its ongoing and future operations, the development, deployment and exploitation of autonomy is being touted as a fundamental means of expanding our operational capabilities. While the promises of autonomy are quite pervasive, the effective development and integration of usable autonomy in military systems represents a significant technological and organizational challenge.

Although automation has been a staple of both military and industrial application for many decades, early applications of automated systems faced similar challenges. Automation reliability has significantly improved, and continues to advance the ability to adapt to the dynamics of an uncertain and ambiguous operational environment. However, these same technologies, like most technological advances, has resulted in unintended, and often undesirable, consequences. A great deal of research has been dedicated to understanding adaptive automation and the role human-automation coordination plays in calibrating automated systems to the dynamics of situations in complex environments.

The Collaborative Interfaces and Teaming Branch of the 711th Human Performance Wing initiated a program to investigate issues associated with advanced human autonomy interaction, particularly as compares and contrasts with existing research on human-automation coordination. The Course Of Action Tools for Decision Superiority (COATDS) task was initiated to investigate the issue of human-autonomy interaction, understand the principles upon which autonomous systems are developed, and leverage this understanding to guide the development of interfaces that would support and enhance human-autonomy coordination in complex work domains.

This final report summarizes the overall literature review portion of the COATDS effort. (All research activities are documented elsewhere.) Additional information on the literature review effort is also available in two earlier publications (Hooper, et al., 2015; Duffy, et al., 2016). The

primary objective of this report is to provide readers with valuable information needed for a working knowledge of the algorithms and techniques and best practices developed to date for their application in an integrated human-autonomy system. Unless otherwise cited, the information is drawn from either commonly accepted knowledge or the writings in the leading textbook for artificial intelligence, “Artificial Intelligence: A Modern Approach” (Russell & Norvig, 2003). Since the intent of this section is to gather and present pertinent information in order for the reader to gain a cursory understanding of the techniques, many examples are taken directly from this text without explicit citations to promote readability. Moreover, the review of methods is couched in the taxonomy we developed for this effort including introductory material that defines key terms and presents a unifying framework and descriptors. The report concludes with short analytical summaries for many problem-solving approaches to provide a convenient reference tool with which interface developers can quickly retrieve key information.

2.0 ARTIFICIAL INTELLIGENCE (AI)/AUTONOMOUS SYSTEMS LITERATURE REVIEW

The COATDS research domain was in mission and route planning as it pertains to human-autonomy and supervisory control systems in an effort to reduce the number of operators needed to control one or more unmanned vehicles. To research this domain we searched for keywords and phrases in many databases to include unmanned vehicles, IEEE Xplore, Engineering Village, Google Scholar, ScienceDirect, SpringerLink, AI conference proceeding, etc. Even though most of the COATDS experimental research dealt with mission and route planning, our literature review covered a wide area. Specifically, the review established the current state of interest and research in utilizing multiple autonomous systems in a human supervisory or teaming control domain.

The focus and process of the literature review shifted with the findings and the progress of the project. This initial stage of the COATDS project included a tutorial series to educate human factors engineering professionals of various fundamental AI techniques that are used within autonomous systems. Therefore, after initial information gathering, the literature review shifted to focus on the AI techniques that would be covered in the next tutorial session. The focus of the AI technique reviews were two-fold: 1) to gather the appropriate amount of demonstrations and explanations of the techniques to educate a non-AI audience of the uses, benefits, and constraints of the techniques; 2) to research how these techniques are used in the COATDS domain. The evolution of the literature review can be summarized in the following steps:

1. Review current government guidance and goals for autonomous systems (e.g., DoD, Air Force, Air Force Research Laboratory (AFRL))
2. Tree-Based search techniques and their uses in mission and route planning
3. State-Space planners and their uses in mission and route planning
4. Probabilistic planners and their uses in mission and route planning
5. Local solution techniques and their uses in mission and route planning
6. Optimization techniques and their uses in mission and route planning
7. Machine Learning techniques and their uses in mission and route planning

Through the course of effort, we gathered over 269 (reviewed and tagged 234) publications (articles), presentations, reports, dissertations, theses, books, and tutorials/class material (see Table 1). The document publication dates ranged from 1959 to 2015 (the year this review was completed before the effort switched to emphasize experimental research). However, approximately 80% of the documents reviewed were published within the last 10 years (2005-2015). Table 1 gives a breakdown of the types and number of documents gathered and reviewed. The 35 documents that were gathered but not reviewed were given a preliminary review, considered pertinent to the research and queued for review, but were not reviewed and tagged. Additionally, there were 39 documents not counted in that were gathered as potentially pertinent to our research but were never given a preliminary review to determine if a full review is warranted. Lastly, there were over 50 documents that were reviewed and deemed not-pertinent to our research. There is no recording of these documents.

Table 1. Literature Type and Number Gathered and Reviewed

Type	Gathered	Reviewed
Publication/Article	209	177
Presentation	29	29
Reports	12	11
Dissertation	4	3
Thesis	6	6
Books	6	5
Tutorial/Class Material	3	3
Total	269	234

Figure 2 show screenshots of a portion of an exported spreadsheet from our research library data and Figure 4 shows a zoomed in screenshot displaying the type key and basic layout of the spreadsheet. The spreadsheet contains the list of documents (and the link to the local file if available) and the document type, publication date, and the associated tags for each document. If we conducted a preliminary review and deemed a document pertinent to our research, we added it to our research library and labelled it with a 'Q' in front of its type and recorded the publication date. After we placed a number of documents in the "Queued for Review" category, we reviewed the abstract and tagged the document based on the abstract to prioritize the full review order. These documents have an 'A' before the document type and indicate that the tags are based on the abstract review and not a full review. These articles were pertinent to the research but were not in complete alignment with the current focus; however, they were anticipated to be of interest further into the research. Currently, there are 45 documents queued for full review in the research library, but they have been tagged based on the abstract review. Therefore, these documents are included in the reviewed column of Table 1. If a document was deemed not pertinent during any time of the review process, it was either not entered into the library or removed from the library. When the document received a full review, it was tagged with the appropriate tags.

We tagged each document with one or more tags from a list of 45 tags. We generated this list of tags based on our research interests, which was dependent on the phase of the literature review, and topics that stood out as important to our research when encountered during the review. For example, the “Technique-Task Pairing” tag was generated when Bruce Clough’s article (Clough, 2002) clearly identified its importance when he stated “As implementers of autonomous control technology, we are not enamored with any particular technique or architecture, we just want the technology that's right for the task”. Because the tags were generated somewhat dynamically during the literature review, some tags were not used to assess a document if the tag had not been generated yet. However, if the tag was already generated, the tag was used to assess the document being reviewed. Due to the evolution of the literature review, it is unlikely that many documents have missing tags. The tags, the tag descriptions, and the number of documents tagged with that description is shown in Table 2.

When tagging a document with one of more of the tags in Table 2, we made a distinction of whether the tag is explicit or implicit. This was necessary to establish which documents explicitly identified to the tag, or else the concepts that they were discussing were closely related to the tag, but the tag was never explicitly stated. For example, if a document talked about the difference between automation and autonomous systems but never attempted to define them explicitly, then we marked it as implicit for the “Autonomous vs. Automatic” tag. Of the 1,171 total tags assigned to the 269 documents, 135 of the tags were labeled as implicit (~11.5%) in 51 documents (~19%).

2.1 Findings

A total of 169 documents were tagged with Allocation, Human-Supervisory Control, Human-Machine Teaming, Planning – Mission, and/or Planning – Path. These are all tags that fit directly with our research domain for the COATDS project. The majority of documents (116) have examples (Academic/Theoretical, Experimental, or Implemented) to illustrate the discussed topics, where 93 of them come from the 169 specifically tagged with our domain keywords. Many of the documents are listed in Appendix A that provides a Bibliography.

As stated above, Bruce Clough’s article (Clough, 2002) established that the appropriate autonomy technology to use to accomplish a task is very dependent upon the task requirements and domain. Corresponding to this emphasis on the task, 72 documents were tagged with “Task Characteristics”, “Technique Comparison”, or “Technique-Task Pairing”. In these documents, we found that similar techniques can satisfy many general task characteristics (e.g., path planning, road networks traversal, resource allocation, optimal, satisficing, etc.). However the technique used (or deemed to be the best to use) for a specific task was tightly coupled with the low-level task characteristics (e.g., reactive to real-time changes in the environment, account for fuel usage, must not transmit communications when near target, etc.). Given these are research papers, their authors are often comparing techniques to reinforce their stance that their preferred technique is the best for a specific domain/problem type, or they are showing where other techniques fail. Often times, these comparisons are made to show different capabilities of techniques for multiple variants or extensions of a fundamental technique (e.g., Rapidly-Exploring Random Trees (RRT) and RRT*Fixed Node (RRT*FN) comparisons from Adiyatov and Varol (2013)). Additionally, the focus for these technique/task characteristics and

comparisons is often on full autonomy implementations and not on the interactions with human operators or supervisors. In the occasions that articles focus on the human interaction with the techniques or system, they do so with an implied assumption of trust. Documents that describe very specific techniques and tasks typically do not discuss what is made available to the user so the user understands what is happening, how the system decided on a solution, and how the system can be controlled to accomplish specific goals. When user interaction is discussed, the literature typically addresses the effectiveness of the interface in the human factors context of the human interacting with the human machine interface (HMI) and not the interaction with the system/agents itself. This, coupled with the above evidence that techniques are extended or altered to fit the specific task or problem domain, renders the notion of developing HMI design patterns to match different AI techniques impractical.

We reviewed 168 documents that addressed specific technique descriptions or implementation within our domain or the implementation in general. Based on these documents, we discovered that the techniques are rarely used in their fundamental form or without combining with other techniques. Typically, researchers will create a variant or extension of the fundamental techniques to fit their domain or satisfy their objectives. Additionally, these variants and extensions are often times combined with other techniques, variants or extensions.

One of the most significant findings in the literature review was uncovered in the documents that described the “capabilities” of autonomous systems and their usefulness for human operators. In these descriptions, the authors defined or used terms to communicate the desired effects. We tagged 61 different documents that used these terms to communicate their ideas. Of these, 33 were tagged with Agility/Robustness, Directability, and/or Transparency/Observability. In these documents, they either spoke directly of these terms or implied them as requirements/capabilities of the autonomous system. Additionally, 28 documents were tagged because they defined the terms satisficing, agent, autonomous, and/or automatic. The ambiguous and/or inconsistent use of these terms in these 61 documents drove us to create a taxonomy and set of definitions discussed in Section 4.0. We felt the collective of these documents were pointing to a common set of goals and descriptions, but the formality of what they were trying to describe was lacking. Therefore, we defined them for the purpose of creating a formal, common language for our program and the three tutorial sessions that we conducted.

The taxonomy that we describe in Section 0, defines the terms Agility, Directability, Observability, and Transparency (ADOT) to serve as a common language in describing autonomous agent capabilities or user requirements. The data in Figure 3 further illustrates our need for the taxonomy. The rows contain the tags that we felt represent a set of foundational techniques used for most autonomous systems where each autonomous system would use a subset of these techniques (or extensions/variants). This figure shows the number of documents that contain tags for each technique when the document also has the specific tag identified by the column heading. For instance, of the 43 documents that are tagged with “Allocation,” just 4 of them also had the “A*” tag. There are many references to these foundational techniques when the documents are describing an agent’s capabilities (i.e., allocation, mission planning, and/or path planning). However, for the documents that describe high-level requirements or intended use (i.e., Agility/Robustness, Directability, Transparency/Observability, and/or Human-Supervisory Control), the foundational techniques are rarely addressed. This further supports the

need to connect these terms to the actual capabilities of the agent, which are defined by the underlying techniques used to develop the agent.

Table 2. Tags Used During The Literature Review

Tag Name	Tag Description: Discusses/Defines/	Docs Tagged
A*	A* and its extensions/variants	30
ACO	Ant Colony Optimization and its extensions/variants	12
Agent Definition	word "Agent" within document	7
Agility/Robustness	terms that relate to our definition of "Agility"	10
Allocation	Task/Resource Allocation	43
Artificial Neural Networks (ANN)	implementation of ANNs or their use as a solution for a problem	22
Autonomous vs. Automatic	either term or the discussion of levels of autonomy	25
Bayesian Networks	Bayes Nets to include Bayesian Filters	22
Connection Based Searches	graphs and trees searches such as breadth-first and depth-first search (excludes A* and RRT)	51
Decision Making	decision making techniques or techniques to help with decision making	42
Directability	the term or desired results of directability	19
Examples - Academic/Theoretical	theories or toy problems to discuss topics	63
Examples - Experimental	experimental examples to discuss topics	42
Examples - Implemented	implemented/applied examples	24
Fuzzy Logic	topic of fuzzy logic	11
Game Theory	topic of game theory	7
Genetic / Evolutionary Algorithms	implementation of GA/EA or their use as a solution for a problem	31
Global Workspace	global workspace and includes reliability and renewal theory	4
Goal-Based Decision Making	topic of goal-based decision making	10
Human Supervisory Control	human supervisory control systems (also used for human-agent interactions)	68
Human-Machine Teaming	human-machine (any autonomous/agent system) teaming	2
Information Theory	topic of information theory	2

Table 2. Tags Used During The Literature Review (cont.)

Tag Name	Tag Description: Discusses/Defines/	Docs Tagged
Kalman Filters	use and/or implementation of Kalman Filters	3
Learning	machine learning (reinforcement, motivated, active, actor-critic, random forest, etc. (not ANNs)	45
Mapping	mapping techniques to include Simultaneous Localization & Mapping (SLAM)	14
MDP	Markov Decision Process (MDP) and Markov Models (includes partially-observable MDPs)	32
Operator / Unmanned Ratio	topic of the ratio or techniques to address the ratio	28
Pattern Recognition	pattern recognition techniques to include clustering	16
Planning - Mission	mission planning techniques, issues, objectives, etc.	55
Planning - Path	path planning techniques	83
Potential Fields	potential field techniques to include vector, force and velocity fields	25
Queued For Review	needs to be reviewed (used for easy filtering)	80
Robust Optimization	optimization techniques to include linear, dynamic, quadratic, and geometric programming	31
Satisficing	the term or concept of satisficing	16
Simulated Annealing	implementation of simulated annealing or its use as a solution for a problem	13
State Assessment	state representations and use (Model Use, World Models, abstract representations, etc.)	19
State-Based Planning	state-based planning techniques to include Stanford Research Institute Problem Solver (STRIPS), Partial Order Planner (POP), (Planning Domain Definition Language (PDDL), etc.	11
Swarms	multiple agent swarms and NOT algorithmic swarms like Ant Colony Optimization	20
Task Characteristics	characterizing tasks for allocation or technique implementation	11
Task Models	use of Task Models	4
Technique Comparison	different techniques and their performance for specific tasks	61
Technique-Task Pairing	different techniques to solve similar problems and selecting a technique	19
Transparency / Observability	term Transparency or Observability or both	30
Trust	topic of human trust in an autonomous agent	5
Value of information	value of information topic	3

The columns that identify with “Capabilities” shed some light on how the fundamental techniques are used within our problem domain. The figure shows that there is a consistent spread of fundamental techniques used in allocation, mission planning, and path planning domains. This is expected since most autonomous systems that humans may interact with are mobile and the paths/routes they take are used in allocation and mission planning. Therefore, these techniques have vast research in their use for very specific problems that fit these high-level domains. However, Figure 3 shows that the fundamental techniques are not explicitly identified as much when describing allocation capabilities. Allocation agents typically generate solutions based on constraints. The fundamental techniques may feed the data used to apply the constraints, but the specifics on how the allocation agent receives its data are usually generalized in these documents. Moreover, this is where the connections between the lower-level agents and the complex agents becomes blurred. When systems become more complex and satisfy high-level abstract problems, the connections to the underlying agents are assumed and not considered in the high-level designs.

The most significant finding resulting from the literature review is that there is a clear division in how two communities are not effectively coordinating to satisfy the users’ needs: communities that develop AI techniques and autonomous systems and communities that design the HMI. The autonomous system communities tend to view their autonomy to be self-contained and only address the user’s input on very abstract level items like “enter a target location”. The human factors designers focus on the displays, controls, and communication protocols that enable the human to interface with the autonomous system, without a full appreciation of the mechanisms inherent in autonomous system methods and their corresponding abilities to support interface techniques for effective human-autonomy coordination. Much of the research pursued during this effort was directed at bridging these gulfs, establishing a taxonomy of human-autonomy interaction that would support more informed interaction between the communities.

Keyword	Capabilities			Requirements and/or Intended Use			
	Allocation	Planning - Mission	Planning - Path	Directability	Agility/Robustness	Transparency/Observability	Human-Supervisory Control
A*	4	6	25	0	0	0	1
ACO	6	8	10	1	0	1	1
Artificial Neural Networks	2	6	13	0	0	0	0
Bayesian Networks	3	7	9	2	1	2	4
Connection Based Searches	5	10	30	0	0	0	3
Genetic / Evolutionary Algorithms	9	12	14	1	0	1	3
Learning	7	8	16	3	1	3	11
Mapping	1	3	10	0	0	0	0
MDP	7	10	18	0	0	0	5
Pattern Recognition	3	6	7	0	0	0	5
Potential Fields	3	6	18	0	0	1	3
Robust Optimization	8	11	15	2	1	3	8
Simulated Annealing	4	6	12	0	0	0	0
State-Based Planning	5	9	4	0	0	1	3
Swarms	7	6	12	1	0	2	4
# Row Tags Assigned for Column Category	74	114	213	10	3	14	51
# Documents With Column Tag	43	55	83	19	10	30	68

Figure 3. Agent Capabilities vs. Agent Requirements/Intended Use and Implementation

3.0 TUTORIAL SESSIONS

During the literature review, we examined different AI techniques and algorithms. While a significant portion of our review was on path planning and task/resource allocation methods, our problem domain of humans managing multiple autonomous systems also involves monitoring and maintenance of execution and dynamic/adaptive mission planning. Since we were looking at basic AI techniques in these areas, we developed three, six-hour tutorials that consisted of very high-level introductions to these techniques, how they are used within our domain, and how they fit within the taxonomy described in Section 4.0. In total, 16 AI techniques or concepts were presented during the tutorials.

1. Tutorial I – Tree-Based searches: breadth-first search, depth-first search, A* (pronounced A-star) search, Dijkstra’s algorithm, and rapidly-exploring random trees.
2. Tutorial II – State-Space Planner, Uncertainty and Local Solutions: Partial-Order-Planners (POP), Markov Decision Process (MDP), Partially Observable MDP (POMDP), Potential Fields and Auctioning
3. Tutorial III – Optimization and Learning: Genetic Algorithms (GA), Ant Colony Optimization (AOC), Simulated Annealing, multiple Reinforcement Learning (RL) techniques, and Artificial Neural Networks (ANN).

These three tutorials included over 70 toy-problem demonstrations that were developed in-house or adapted from open-source programs and videos. These demonstrations were used to illustrate the functionality and capability of the algorithms and techniques.

The goal of the tutorials was to give HMI designers a cursory understanding of some basic AI building blocks that agent developer use when creating autonomous systems. The belief was that with knowledge of the underlying philosophies, human factor specialists would have a better understanding of what can and cannot be accomplished when designing interfaces aligned with the agents. These tutorials covered many techniques that the HMI designers frequently hear mentioned when agents are being described. The goal was to provide HMI developers a better understanding to know which questions to ask that help address interface requirements.

While working on the tutorial material and conducting the literature review, we discovered terminology differences. This prompted us to develop working definitions for these terms to be used in the tutorials. The result was the agent descriptor taxonomy discussed in Section 4.0.

4.0 AGENT DESCRIPTORS

Key limitations in the strength and usability of a specific HMI can be addressed through an improved dialog between agent developers and HMI designers. Communication between the two design communities is critical to developing systems that support appropriate levels of human-autonomy coordination. To improve the dialog, it is of benefit to consider establishing a common language with which HMI designers and agent developers can communicate certain goals and features. For the HMI designers, the language's descriptors communicate desired agent capabilities. The same descriptors can help scope agent development, such that the agent's capabilities are tuned to support interface features. Thus, a common language/taxonomy that aids productive dialog between agent developers and HMI designers should also enable HMI designers to construct more effective interfaces and agent developers to have well-scoped feature sets to support.

Through our research, we considered a number of agent development techniques in the context of designing more effective HMIs. The long-term goal of this research initiative was to support system developments such that future complex Air Force operations benefit from the joint capabilities of advanced agent and HMI technologies. In doing so, we considered the various affordances and constraints of a number of techniques. In evaluating these, we found three key issues:

- 1) Terms used to describe agents by various communities: HMI, agent developers, and others (such as from the DoD) rarely overlap;
- 2) When the terms did overlap, there was a distinct mismatch in the meaning of the terms between the communities; and
- 3) Lack of joint understanding can cause a disconnect between HMI designers and agent developers of new technologies for autonomous systems.

The lack of a common language to describe desired goals and capabilities often leads to limitations in the resulting system's capabilities. Even with communication between the agent developers and HMI designers, HMI designers may make feature requests that are considered difficult or infeasible by the agent developers. The absence of a common taxonomy can also affect agent development, as the agent developers, unmindful of the potential HMI concerns or desired capabilities, may develop products that do not support many usability-related features. The HMI in some systems can be unmanageably complex if designed only from the perspective of the autonomous agent system rather than the joint operator-autonomous agent system (Bartram & Ovans, 1995; Steinfeld, 2004).

In an effort to alleviate disconnect between agent descriptions and the desired features of the human interface, we focused on terms that are familiar to HMI designers as they relate to an agent's capabilities. However, even within the human factors and cognitive psychology literature, the same terms are employed to describe different agent features in the context of human interaction and understanding. There is very little consistency between publications describing desired autonomous agent features (Masiello, 2013; Dahm, 2011; Murphy and Shields, 2012), so in order to establish a common language, consistent definitions are needed (Burke et al., 2004; Korsah, Stentz & Diaz, 2013).

The descriptors that we propose help to describe an autonomous agent through the perspective of an external observer or operator. An agent is a decision maker, program, vehicle, operator, system, or some other entity that can affect the environment. Our definition of an agent is:

Agent –An agent is anything that can be viewed as perceiving its environment through any means (e.g., sensors or inputs) and able to effect change within that environment.

Per the definition of an agent, an external observer could be a human or could be another agent. Many human supervisors of autonomous systems want to understand the what, how, and why of the behavior of autonomous system(s). They require this information to determine if the mission is on track, an error has occurred, intervention is needed, a decision should be made, etc. In general, the human supervisor may not have all the information to come to these conclusions. Therefore, they need enough satisficing information to make decisions on the missions they are supervising.

Satisficing – A potentially suboptimal solution or decision, generated with incomplete data that will still accomplish the task and is acceptable to an operator.

In order to obtain this satisficing information about the autonomous agent, the operator requires information from the autonomous agent. This information includes items that inform the supervisor of what it is doing, why it is doing it, how it decided to do it, and what the supervisor can do to influence the agent's actions/behaviors/decisions. To describe these agent abilities or the requirements of the supervisor/observer, we developed four descriptors that help to communicate and understand what the agent can provide. These terms are: Agility, Directability, Observability, and Transparency.

The remainder of this section presents the results of our efforts to bridge this communication gap between the agent and HMI communities. First, we introduce descriptors and provide discernable definitions for each. These descriptors are the basis of our proposed taxonomy. Next, we define levels within each of the defined taxonomic descriptors that provide a more granular description, supporting better communications between agent and HMI communities. To augment the level descriptions, we provide examples of level assessments using the techniques and methods covered in the tutorial series. Finally, we apply the proposed taxonomy to example systems to show how the descriptors help to close the loop between HMI designers and agent developers so that their joint developments can enhance operator-autonomy coordination.

4.1 Descriptor Definitions

4.1.1 Agility Definition

The first descriptor is agility. Other authors discuss agility by inverting the concept. They call this brittleness (Clare et al., 2012; DePass et al., 2011; Klein et al., 2004; Christoffersen & Woods, 2002; Woods & Hollnagel, 2006). “Inevitably, robot capabilities will exhibit brittleness as situations develop beyond their boundary conditions... these represent challenges to the adaptive power or resilience...” of the robot and human-robot team (Woods & Hollnagel, 2006). If a design is brittle, it results in unnecessary performance trade-offs and “additional manpower, vulnerabilities and lack of adaptability for new missions” (Murphy & Shields, 2012). Although there were plenty of references and examples of brittleness in the literature, we will use a

positive characteristic in describing the system (i.e., agility) vice the negative effect of not having the capability (i.e., brittleness).

Agility – the ability of an agent to respond in an effective manner to system dynamics (e.g., environmental changes, operator input, situational dynamics, etc.) within a short timeframe.

The key lesson related to agility is that the terms agility, robustness, resiliency, adaptability, and flexibility are not the same thing, though they are related. Agility is the ability of the system to leverage one or more of these characteristics to adjust the system's means of executing the mission, and do so quickly.

Robustness – The ability of a system to maintain operational viability in the face of failure or extenuating circumstances.

Flexibility – The ability of a system to conform to its environment and operational parameters.

Resiliency – The ability of a system to restore its operational viability after it has been compromised.

Adaptability – The ability of a system to change in response to operational parameters or environmental pressures.

The delineation between the above characteristics (robustness, flexibility, resiliency, and adaptability) and agility is that a system may possess any or all of these characteristics, but not be agile. However, if the system is agile, it does possess at least one of those characteristics. The difference is based on time. All of the characteristics can contribute to agility, but unless it can engage that characteristic 'quickly', the system is not agile. The timeframe that drives agility is subjective, and depends on the operator, the domain/environment, and the system itself. This is intentional, since it is the operator and/or system developers who define at what level agility is achieved. The "short timeframe" means that the associated parties consider the system to have responded quickly or within the required timeframe. There is no globally attributable timeframe for agility.

4.1.2 Directability Definition.

Directability discussions mainly focus on how an external agent can influence the behaviors, actions, or decisions of another agent (Morley & Myers, 2001; Klein et al., 2004; Christoffersen & Woods, 2002; Truxler et al., 2012). Our definition for directability is:

Directability – The ability for an external party to influence a change in the operation of another agent in order to accomplish a specific output or objective.

Support for directability is necessary because of limits on agility. If systems had unlimited capacity to adapt to change as well as unlimited awareness of the need to adapt, directability may not be necessary. However, limits on adaptability and awareness are inherent in all systems. In considering directability, one must ask the questions: "How much directability can we have and how much is needed?" These questions are necessary in defining the requirements for the system and defining the task breakdown of the agent with respect to directability (i.e., how much control the user should have). In answering these questions on directability, it becomes clear that directability can be at odds with autonomy and may require adjustment based on the desired

level of autonomy. However, directability is inherent in our vehicle control application, so it is important to know what is directable in order to provide operators with appropriate interfaces.

4.1.3 Observability Definition.

Observability enables “users to enter information, review default assumptions, and inspect and compare alternative “courses of action” (DePass et al., 2011). In other words, observability is the ability for an observer to measure the outputs of the system and understand WHAT the agent has done, is doing, or will do soon. Our definition for observability is:

Observability – The level to which the exact state of the system can be determined from measurements of the output over a finite time.

4.1.4 Transparency Definition.

Authors typically do not define the word transparency explicitly when they use it, but imply its meaning in the context of the scenario they are describing or what transparency provides. “Machine transparency enables the human to understand what the machine is doing and [WHY]” (Masiello, 2013). For the user to understand what the agent is doing and why, the agent “must be able to make pertinent aspects of their status and intentions obvious to their teammates” (Klein et al., 2004). In addition to the “why” of an agent’s actions, some authors describe transparency as the insight or understanding of HOW the agent generated a solution, plan, action, etc. (Clare et al., 2012). Therefore, we work with the following as the definition of transparency:

Transparency – The ability to provide information about why and/or how an agent is in its current state.

In more general terms, transparency is the understanding, or knowing, of what the agent is “thinking”. The Defense Science Board’s Task Force Report on the Role of Autonomy in DoD (Department of Defense) Systems implies this general meaning of transparency in its recommendation for the future development of autonomy: “The objective should be to create a technology base of diverse, platform-independent, transparent cognitive functions and tactics for integration into new missions” (Murphy & Shields, 2012).

When using the terms transparency and observability, some authors have described either transparency as our definition of observability or observability as our definition of transparency. For example, Christofferson and Woods (2002) define observability as “opening up the black box”. Additionally, Bass et al. implied that observability gives the user insight into “the automation’s judgment strategy” and, in turn, “found that human judgment performance improved when meta-information was provided regarding how the automation integrated input data to derive its judgment ... compared to when only the judgment was provided” (Bass, Baumgart, & Shepley, 2013). These descriptions match more closely to our definition of *transparency*, as opposed to *observability*, since it is giving the operator more insight into why or how the agent is operating, not what its state happens to be at a given time. In contrast, it can be viewed that (Olsen & Wuennenberg, 2011) merge our definitions of observability and transparency when they describe visibility on a user interface.

Having more insight into why and how a system is operating helps the human operator anticipate agent behavior and calibrate one’s trust in the agent’s functionality. “For humans and machines to function as an effective team, there must be an understanding of and confidence in its behaviors and decision making across a range of conditions. If the robot is transparent to the

user, than only one mental model is required. Thus, transparency is a desired element of efficient interaction” (Goodrich & Olsen, 2003). “This understanding will enable an appropriate level of reliance on, or trust in, each team member for a given situation” (Masiello, 2013). When a user has trust in the system, then they will have more confidence when making decisions and directing the agent to perform its autonomous missions.

A simple way to remember the differences between transparency and observability is to consider what question it answers. If asking *what* a system is doing, observability will drive the accuracy of the answer. If asking *why* or *how* a system is doing something, then transparency will be the primary factor in the accuracy of the answer.

4.2 Descriptor Levels

The descriptors defined in Section 4.0 help to communicate the characteristics of autonomous agents in a general, overall sense. However, identifying a level or quality of each of these characterizing descriptors gives the HMI designer a better understanding of the agent’s abilities. The following sections identify the levels that can be assigned to each descriptor and what criteria pertain to the assignment of each level. An important item to note is that an agent can fit within multiple levels dependent upon the situation. When this occurs, the dependencies that change the level should be identified when describing the descriptor and levels for the agent.

Each of the techniques covered in the tutorials were analyzed for each descriptor and given one or more level ratings. The assignment and discussion of ratings for some techniques are used to illustrate how the descriptor levels can be assigned to these basic techniques. Section 0 will demonstrate how we can analyze hybrid systems (systems that use multiple techniques and methods) and what modifications can be used to change the levels. Although we may not discuss the analysis of all tutorial techniques for each descriptor in this section, Appendix B contains the combined summary of these analyses for each tutorial technique along with a summary of the technique. When determining the levels, the scope must remain within the context of the domain. In other words, an agent may be defined as highly agile assuming the domain is a cold, dry environment. This does not mean the same agent is highly agile in a hot, humid environment. Therefore, our analysis of each AI technique is for the domain of path planning and resource allocation. Note, these analyses are subjective reflecting the perspective of an agent developer describing how an interface designer can use the techniques/agents based on what is possible.

4.2.1 Agility Levels.

As defined in Section 4.1.1., agility is the ability of a system to respond in an effective manner to new inputs within some perceived “short time.” The levels of agility are: none, minimal, limited, moderate, and high.

None: if an agent cannot respond to changes. Also assigned if the agent cannot adapt to a change without restarting, as if from a clean slate. That is, it cannot leverage the existing state nor its existing knowledge to handle a change.

Minimal: if the agent can be changed from its general implementation/configuration to handle very specific changes, but otherwise performs as if its agility rating is **None**.

Limited: if the agent’s current implementation can respond to changes for very specific cases.

Moderate: if there are situations where the agent can generally respond to changes, but there are also situations where the agent cannot respond to those same changes.

High: if the agent, under most situations, can consistently handle changes.

When assessing agility, the work domain and the actual implementation of the algorithm must be taken into consideration. Algorithms like Breadth-First Search (BFS), Depth-First Search (DFS) and Dijkstra's algorithm require for the environment (i.e., the state it uses to calculate a solution) to be static. If any of the state data changes, the algorithm must start over since the previous solution may no longer represents the current environment. Likewise, MDP require static information for transition models, reward functions, discounting and available actions. If any of these items change, the MDP values must be recalculated and the policy re-evaluated. Therefore, MDPs are not agile.

In their most general implementations, the A* and RRT algorithms cannot react to changes within the search tree. If there is a change within the environment, the search must start over since the previous tree no longer represents the current environment. Therefore, the RRT and A* are not agile. However, there are variants with each of these algorithms that allow for real-time updates and searching of small subsections of the state space when they change (e.g., Dynamic A* (D*)). If implementing these variants, A* and RRT can be assessed minimal agility.

Techniques like POPs and partially-observable MDPs (POMDP) can be assessed with limited agility. POPs and POMDPs can adapt to the environment during execution because they use the current state to determine what action to engage next. However, the overall plan or policy would need to be regenerated if any states or actions change.

Simulated annealing runs on a schedule, which defines its progress through a search. Without the ability to reset this schedule, there may be moderate agility if the minimum the algorithm is working towards is not significantly different (in terms of search space shape) from the new one. If the ability to reset the schedule is added, then it may become more agile, but not highly agile. It can respond to changes in the search space, even when a search is in progress. Similarly, Ant Colony Optimization (ACO) is moderately agile. If a new target point is selected, the ants will pursue that target. Some time may be spent waiting for pheromones to evaporate, but ACO will generate new routes. Obstacles and other such aspects are handled dynamically, since the pheromone trail cannot be followed and that portion of the route will evaporate.

A number of techniques can be assessed as High agility. Localization methods are highly agile provided the state space is known. Since the strength of this method is to estimate its current location based on sensor inputs, the method can account for changes in the environment (e.g., moving objects) as long as the environment in which it is localizing remains static. In other words, the moving objects can be incorporated as part of the noise in the sensors, but the map (e.g., walls, corridors, gaps, etc.) must be static in order to manage the uncertainty in the location estimations. Potential fields give an agent a reactive behavior to the environment. Implementing potential fields in an agent gives the agent a high level of agility in respect to changes or obstacles in the environment. No a priori knowledge of an obstacle's (static or dynamic) location is required since obstacles are handled upon discovery. Therefore, the localized representation supports many other environmental changes without the need to plan extensively. Provided an auctioneer is made aware of it, auction methods can accept new agents and new tasks. However,

there is the potential for a lower quality overall allocation when allocating a number of tasks at one time.

Since Reinforcement Learning (RL) requires an agent to learn within the environment through experiences and feedback, the agent will learn the new environment if it changes. This gives RL techniques a high level of agility.

Finally, dependent upon the implementation and the environment, Artificial Neural Networks (ANN) can be assessed at any level of agility. The flexibility and diversity of ANN representations are both their greatest strength and greatest weakness. Since some network structure can exist for any continuous function, ANNs can solve an extensive set of problems. The ability to restructure the ANN and for it to learn from new information could provide a critical level of agility that cannot be achieved by any other known method. Additionally, if the network is underfit or overfit, its agility may be severely reduced or non-existent. Otherwise, ANNs can be highly agile. New training data may change some weights, which change the resulting output. Depending on the domain, the ANN may already handle the new data without training.

4.2.2 Directability Levels

Directability, as described in Section 4.1.2, is the ability for an external agent or source to influence a change in the operation of another agent in order to accomplish a specific output or objective. The levels of directability are: none, minimal, limited, moderate, and complete.

None: if the agent provides no ability for the human operator to achieve a targeted output via changing an agent's inputs ("inputs" include changes to variables/techniques in the agent's computational process).

Minimal: if the operator can cause changes to some agent outputs, decisions, or actions via changing inputs, but in a way that may not reflect the operator's desired output.

Limited: if the operator can manipulate a subset of the agent's inputs to change a subset of the agent's outputs to reflect the operator's desired output.

Moderate: if the operator can make the agent achieve desired outputs via manipulation of the agent's inputs, but only indirectly.

Complete: if the user can make the agent achieve desired outputs via manipulation of the agent's inputs directly.

Many of the basic search algorithms like BFS, DFS, Dijkstra's and A* are implemented for the specific task that they are providing a solution and therefore, will offer no directability. The algorithm runs as expected and will return the same solution, on the same search tree, every time. However, many times these algorithms are used in conjunction with others or are implemented to give minimal directability for a user. For example, A* typically works with just one heuristic to generate the optimal path based on the cost function (e.g., distance), but if the user is given choices to select different cost functions (e.g., time, fuel, etc.), then the algorithm may be assessed as minimal or limited directability dependent upon the user's overall needs.

Minimal directability is assessed when a user can try new "settings" but may not get the desired output they seek. Simulated annealing, ACO, GAs fall in the minimal level since they may allow the user to influence the search state but cannot guarantee a specific solution. Implementations

of these algorithms may allow the operator the ability to seed the algorithm to direct the algorithms to intended solutions. However, the stochastic nature and randomness of the algorithms make no guarantees that the operator's influence will have the desired effect on the final solution.

Many algorithms we examined fell in the limited directability level. These algorithms allow the user to change values that will ultimately change the output to better reflect the operator's intentions. For example, POPs may allow the user to set priorities on actions or goals, which can dictate which actions are accomplished first in the plan. Localization techniques may allow the user to provide the exact location or region of the agent's current focus to help it localize quicker. Potential field techniques may allow the user to adjust force response values to help with dead ends. Moreover, RRTs may allow the operator to bias towards goals (exploitation), unexplored regions/Voronoi space (exploration), bounded regions, or a combination of all. However, there is no guarantee that these controls will result in the intended direction of the operator since the state is a main factor in the output. These directability options may provide shorter runtimes or more desirable solutions for the operator, but the ability of the algorithm to produce more optimal solutions may be degraded and there is a possibility that the output still does not meet the operator's desires or expectations.

MDPs and POMDPs can be assessed as either limited or moderately directable as long as the domain allows changes within the reward functions, transition models, and discount rates. This can be used as an indirect means to affect the policy by injecting priority in speed to complete or risks of non-optimal outcomes. However, the resulting policy may not resolve to the user's intended policy. Additionally, POMDPs may allow a user to provide a subset of the search space to the agent as the possible, or even the exact location of the agent. However, there is no guarantee that the agent will localize to the correct location. This could happen for a number of reasons: 1) The accuracy/correctness of the sensors may identify the wrong state space. 2) The internal probabilistic models of the sensors are not correct. 3) The same (or very similar) sensor readings are in other parts of the state space, which are equally likely. 4) The state space at this location may not be feature rich enough to localize.

RL techniques can be assessed as completely directable. Since the user has ultimate control over the training data, the user can direct the method on what to learn. Although the specific elements of directability depend on the reinforcement learning technique used, they may allow the user to set the learning rate, the amount of exploration vs. exploitation, and the training that the agent may experience. This allows the user complete directability in the learning process, but it does not guarantee that it will learn the intended behavior because of the stochastic nature of the environment and agent actions.

4.2.3 Observability Levels.

The state of a system is said to be 'observable' if the exact state of the system can be determined from measurements of the output over a finite time (see Section 4.1.3). The levels of observability are: none, minimal, limited, moderate, and complete. Observability and transparency are tightly coupled. Users typically want more observability to increase their knowledge of the system and therefore, increasing the system's transparency.

None: if the agent is a "black box", in which it only shows the outputs and nothing that helps the operator understand what the agent is doing, has done or is going to do.

Minimal: if the agent can show the operator some of what it is doing, but additional state understanding cannot be obtained, even with multiple observations.

Limited: if the agent can show the operator what it is doing, and support requests to obtain more state under-standing, with a number of observations.

Moderate: if the agent can provide enough information on what it is doing, such that, over a series of observations, its state can be completely understood by an operator.

Complete: if the agent can show its complete state to an observer. Note: this does not mean that the internal state is shown, although it is possible. Ideally, to meet the ‘complete’ level, this requirement needs to be communicated before the agent development process begins. Having complete observability typically results in increasing the agent’s level of transparency as well.

We did not assess any of the techniques we examined to be at an observability level of none or minimal. This is mainly due to the perspective that we had while assessing these techniques. We based our assessment on the ability of these algorithms and techniques to make their state or solution observable. Therefore, all of the techniques were capable of making these observable. We did not include agents that could not make their state observable. However, we still defined these two lower levels in case there are systems that do not provide observability or provide minimal observability. For example, assume you are given an agent to use with your system that does not have a GPS or any other localization mechanisms. If we were assessing this agent on observability, we would have to say it has none since it cannot make its location (i.e., its current state) observable to us.

All techniques we researched and assessed could be considered completely observable except for RRTs and GAs. RRTs and GAs were assessed as limited observability. RRT’s observability is limited to the tree it is generating and the final solution. It allows the user to observe what the agent has accomplished, but gives no insight into what it is doing or will do since the selections are random. GAs have limited observability since the state of the GA is the population at any given iteration. As long as the population can be extracted, there is no problem with understanding the state. However, this merely helps to define a percentage of the search completion, not observability into the algorithm’s state.

The high number of techniques evaluated at completely observable is due to the nature of these basic AI techniques. The internal state can be made available at any time as long as the implementation allows for it. Additionally, the observability rating is based on what the operator may want to observe. For example, suppose using a tree-based search technique, one user wants to know the number of nodes currently searched and another wants to see the pending nodes of the search tree. If the number of nodes is provided and just the current state of the searched nodes (i.e., the tree), then one user would assess this as completely observable and the other would assess it as limited or moderate. Due to this subjectivity and the fact that many basic algorithms do not provide observability into everything that can be made observable, we assessed more than half of the techniques in the limited to complete range.

Conversely, there were a number of techniques (e.g., POP, MDP, Localization, Potential Fields, Auctioning, and Simulated Annealing) that were assessed as only completely observable. These techniques were assessed this way and not with a range because we feel that the observability that would be desired in these techniques are typically, or easily provided to give complete observability. For example, even with noisy sensors or poor state estimates, the responding

behavior of the localization method can be shown. Additionally, the force vectors can be made available. The combination of the two shows exactly what is happening regarding the agent and its behavior. Localization methods can provide complete observability if the estimated locations are shown. However, there may be some stochastics in the initial particle estimates (if particle filters are in use), and this may diminish the effectiveness of the observability if there are too many possible locations. If, on the other hand, a few “very good” estimates are present, then the observability provided can be more useful/effective for the observer. Auctioning techniques can make the bids during each bid cycle and final allocations available, which allows the user to understand the state of the auction at any time. For simulated annealing, an operator could view the current best solution, state of the annealing schedule, and the most recently tested solution. This is everything needed to reconstruct the state of the algorithm.

4.2.4. Transparency Levels.

As stated in Section 4.1.4, the transparency is the ability to provide information about why and/or how an agent is in its current state. The levels of transparency are: none, minimal, limited, moderate, and complete. Since transparency is the knowledge of why and how an agent is in its current state, this knowledge can be acquired through developer discussions. Typically, a user’s understanding of the underlying mechanisms within the agent and their current state is what increases transparency. Increases in transparency can come from internal knowledge and increased observability of the agent’s state through time. Therefore, observability and transparency can be tightly coupled in assigning transparency levels. Additionally, one can view transparency as the likelihood of the agent acting/behaving/performing as expected.

None: in cases where the agent is given inputs and generates an output with no straightforward indication on how or why the agent produced the output.

Minimal: if there is no observability into the state of the agent and the limited understanding of the agent’s behavior extends from the operator’s general knowledge of the underlying system.

Limited: if observation and knowledge of the underlying mechanisms is required to determine why and how the agent is in its current state, but there are some elements that may produce unexpected results.

Moderate: if observation and knowledge of the underlying mechanisms is required to determine why and how the agent is in its current state.

Complete: if the internal computations can be made available for a user to understand how and why the system behaves in the way that it does.

Like observability, all techniques that we assessed for the tutorials were classified as limited or above. Again, this was due to our domain and perspective. Also, each of these basic AI techniques have relatively high observability ratings and are well known, especially after the tutorials. Many of the very basic and deterministic techniques were assessed as completely transparent, because knowledge of how the algorithm works, the cost function, vector fields, or the policy gives the user complete understanding of the how and why a solution was selected (e.g., BFS, DFS, A*, POP, MDP, potential fields, etc.).

Simulated annealing, RRT, GA, and ACO were not assessed as completely transparent. This was due to the stochastic nature of these algorithms and the randomness of exploration. A solution’s

selection may be understood by the user, but there are many opportunities for the path to the solution to make unintelligible changes.

The learning and probabilistic techniques had the lowest assessment of limited. This is due to randomness in the underlying selection processes (e.g., localization and GAs), the complexity of edge weight calculations (e.g., reinforcement learning and ANNs), or the quality of the training data that makes something learn one less desirable behavior over a more desirable behavior (e.g., reinforcement learning and ANNs).

Auctioning is moderately to completely transparent when the user is given the methods used to calculate the costs and to make selections. With this information and information on the bids, users can see the reasons an allocation was determined. Barring any unusual allocation or selection options related to the bidding, auctioning is completely transparent. However, auctioning is limited if the user wants to understand why an agent made a particular bid. Most auctioning methods assume that the agents define their utilities and bid so that the results of the auction serve a common objective. Therefore, transparency into the agent utility calculation is not available.

4.3 Taxonomy Summary

Tables 3 - 6 below attempt to characterize each of the techniques in terms of the requisite support functions identified in the framework (i.e., Agility, Directability, Observability and Transparency) as we have defined them. Each technique is graded in terms of the extent to which it can contribute, as described in the previous section, to the framework's support functions (i.e., None, Minimal, Limited, Moderate, High). A more detailed explanation of these characterizations is presented in the technique descriptions in Appendix B.

Table 3. Assessment of Techniques in Terms of Support for System *Agility*

None	Minimal	Limited	Moderate	High/Complete
BFS				
DFS				
Dijkstra				
A*				
RRT				
POP				
MDP				
				Localization
		POMDP		
				Potential Fields
				Auctioning
			Simulated Annealing	
			ACO	
			Genetic Algorithm	
				Reinforcement
ANN				

Table 4. Assessment of Techniques in Terms of Support for System *Directability*

None	Minimal	Limited	Moderate	High/Complete
BFS				
	DFS			
Dijkstra				
		A*		
		RRT		
		POP		
		MDP		
		Localization		
		POMDP		
		Potential Fields		
		Auctioning		
	Simulated Annealing			
	ACO			
	Genetic Algorithm			
				Reinforcement
ANN				

Table 5. Assessment of Techniques in Terms of Support for System *Observability*

None	Minimal	Limited	Moderate	High/Complete
		BFS		
		DFS		
		Dijkstra		
		A*		
		RRT		
				POP
				MDP
				Localization
		POMDP		
			Potential Fields	
				Auctioning
				Simulated Annealing
		ACO		
		Genetic Algorithm		
		Reinforcement		
		ANN		

Table 6. Assessment of Techniques in Terms of Support for System *Transparency*

None	Minimal	Limited	Moderate	High/Complete
				BFS
				DFS
				Dijkstra
				A*
		RRT		
				POP
				MDP
		Localization		
		POMDP		
				Potential Fields
		Auctioning		
		Simulated Annealing		
		ACO		
		Genetic Algorithm		
		Reinforcement		
		ANN		

4.4 Multi-Agent System Analysis

During our literature review we discovered that describing an agent based on the primitive techniques used is not useful when systems employ multiple techniques that are specific to the intended domain and mission. These agents typically use variants or extensions of fundamental techniques and combine them to perform complex tasks and support a more general (or abstract) mission. Additionally, and most commonly, an agent can be the combined effects of multiple agents working independently and/or together. For example, an autonomous car can be viewed as a single agent but it is made up of a breaking agent, an obstacle avoidance agent, a path planning agent, and many other agents to create the capability that the autonomous car provides. The “car” is a single-agent abstraction of the combined effects of the many agents that it employs. Therefore, it can be described noting each of the individual, fundamental techniques described in Section 4.2, or given a general level assessment, a difficult choice as explained next.

For the purpose of this report and to be consistent with our definition of agent, we are calling these single-agent abstractions “multi-agent systems” if their capability/behavior is the combined effects of two or more agents. In this section, we will present case studies to illustrate how the taxonomic descriptions of the agents used within the multi-agent system differ from the taxonomic descriptor of the multi-agent system as a whole. These differences are not always intuitive and do not always correlate in an expected manner (e.g., increasing directability in an underlying agent may decrease the transparency of the multi-agent system).

The following sections give examples of multi-agent systems and sample perspectives for evaluating the taxonomic levels of the system and some of its underlying agents. Then we will describe a change in an underlying agent and assess the effects it has on the overall system. The goal of these case studies is to show that:

- 1) The taxonomy can be used to describe agents at any level of abstraction and is based on the perspective of the person describing the agent and their overall intended use of the agent;
- 2) The taxonomic descriptions of the underlying agents in a multi-agent system are not additive, cumulative, transitive, etc. The effects of changing the taxonomic levels of an underlying agent may have an effect on the taxonomic description of multi-agent system, but the effects must be evaluated to understand the changes in the taxonomic level or the addition/reduction of requirements for the multi-agent system to maintain its taxonomic levels before the change;
- 3) Because of item 2, the notion that HMI design patterns can be developed based on the underlying agent’s capabilities/descriptions is impractical due to intended use, domain, and custom implementation.

4.4.1 Operator-Unmanned Vehicle Control System.

In a tri-service autonomy focused research initiative that AFRL led, an operator-autonomy teaming control system was developed that provides base defense capabilities by enabling effective control of multiple heterogeneous unmanned vehicles including unmanned aerial, ground, and sea surface vehicles (Draper, et al., 2018). The intent of this effort was to illustrate a new agile control approach for collaborating with automation in future UxV applications. This system utilizes decision-making, resource allocation, constraint solving, path planning, and other autonomous agents to assist the human operator in making decisions and executing the current

mission. For example, the system utilizes path planning algorithms that rapidly calculate the most efficient route for a vehicle to take to a specific point while taking into account no fly zones, unpassable terrain, and environmental conditions (see Rasmussen, Kingston, & Humphrey, 2018, for related work). Additionally, a resource allocation agent (Hansen, et al., 2016) recommends which vehicle(s) to assign to a specific task based on the vehicle's probability of success, estimated time enroute to task location, fuel capacity, effects on existing tasks, etc. Lastly, a monitoring agent identifies when a current mission deviation occurs and alerts the operator. The autonomous agents within the system allow the operator to offload manual control of the vehicles to the automated system, allowing more time to focus on assigning high-level (i.e., abstract) tasks, monitoring the situation, and adjusting to unexpected events (Draper, 2007).

Consider the path planning agent that generates a route for an autonomous ground vehicle from the vehicle's current location to a target location using a road network. The HMI provides the operator a "god's eye" view of the domain on a map (Figure 4). The agent development team knows the algorithms and the basic requirements of the domain (e.g., navigate to the target location), and the HMI team knows the agent can perform this navigation. With this most basic knowledge, the HMI is limited to the selection of a location to which the agent navigates.



Figure 4. Sample Ground Route Shown On a Map in Simulation

The agility of the system is based heavily on the algorithm(s) in use. In this example, the agent developer chose Dijkstra's algorithm and built an all-to-all mapping of the domain, performed when the agent starts. Then, all routes are a simple lookup from the existing table. This makes the agent response fast, but it severely affects agility. It prevents the agent from responding to extenuating events, such as blocked or slow roads. Thus, the agility level that this approach provides is None: for each new input (e.g., blocked road), it has to restart Dijkstra's algorithm to solve its route. Additionally, the level of directability it provides is None: the agent cannot visit certain locations that the operator may desire. However, it can have Moderate observability, since the distance metric in use is simple to understand or see in the provided solution. Finally,

transparency is Complete, since the decisions regarding the route selection are easy to understand.

Now, consider the case where HMI designers wish to improve the agent's agility and directability: they want the agent to handle blocked roads and to visit certain locations enroute to its target location. The agent then must be reworked to comply with these new requirements. This rework is better scoped since it is clear what the HMI designers intend: increase both agility and directability to the Limited or Moderate levels, and keep the rest of the levels the same as much as possible. In response, the agent developer changes the routing algorithm to A*. This takes more time to calculate for each request, but it is still sufficiently fast to provide adequate agility for the domain. Given its speed, and its on-demand activity, it improves agility to Limited: it can perform a calculation based on a blocked road, as it is removed from the agent's list of candidate edges. It also improves Directability, since the operator could exclude roads to visit. The improvement in directability that the HMI designer desired was to route the vehicle through certain points. The agent developer then provides a way to chain the target locations, where the agent solves for a route to the first location, then from the first to the second, etc. This helps to realize Complete Directability in defining the points to visit, along with Limited Directability in avoiding certain roads. This additional Limited Directability was not called for by the HMI designers, but has emerged as an additional feature that the agent can provide. The other taxonomic levels remain the same. Thus, with a change in the algorithm and some sacrifice in the runtime, the agent shifted from a precomputed solution to an on-demand solution. This increased agility and directability in a way that was desired by the HMI designers, provided an additional feature that the HMI designers did not anticipate. Given this taxonomy's common language for describing these features in the context of the domain, these improvements were facilitated and attention can then shift to designing interfaces by which the operator communicates more detailed route requests (e.g., which roads are blocked) to the agent.

The above example describes the taxonomic changes of the path planning agent after updating the algorithm it uses and allowing new inputs. Figure 5 shows the comparisons of the taxonomic assessments of the fundamental Dijkstra's algorithm, the implemented Dijkstra's algorithm, and the implemented A* variant of the algorithm. From this figure, we see that the perspective of the assessment makes a difference on the fundamental algorithm and the implemented. The assessment of the fundamental Dijkstra's algorithm (see summary sheets in Appendix B for full assessment) was made looking at the algorithm in general and the ranges of what can be supplied to the operator as it relates to the algorithm's internal state, progress, and final solution and not on a specific domain or implementation. The implemented Dijkstra's algorithm eliminated these ranges since the information it provides and the working domain is known. The taxonomy allows clear communication of how these changes effect the agent as shown in the figure. However, when the changes occur in an agent that is part of a multi-agent system, we must reassess the system at each level of abstraction to understand why, if any, taxonomic descriptions have changed or new/reduced requirements are necessary.

Method	Levels					Descriptor
	None	Minimal	Limited	Moderate	High/Complete	
Dijkstra Fundamental (see Summary Sheets)	Avoid Roads					Agility
	Goal Cutoff					Directability
			Search Progress			Observability
					Deterministic	Transparency
Implemented Dijkstra	Avoid Roads					Agility
	Avoid\Waypoints					Directability
				Distance		Observability
					Shortest Path	Transparency
Implemented A* Variant			Avoid Roads			Agility
			Avoid Roads		Waypoints	Directability
				Distance		Observability
					Shortest Path	Transparency

Figure 5. Path Planning Agent Implementations Compared to Taxonomic Levels of the Fundamental Dijkstra's Algorithm Assessment

The operator-autonomy control system contains a hierarchical architecture that abstracts the mission from the individual tasks of the autonomous agents (e.g., vehicles, planners, allocators, etc.) to high-level goals. There are three layers to this architecture, which includes more agents working in parallel or together the higher up the hierarchy. We show a generalization of the hierarchy in Figure 6. Each layer can be viewed as a multi-agent system and can be analyzed using the taxonomy as a single-agent. However, increasing the agents in a multi-agent system increases the importance of establishing the bounds (e.g., domain, intended use, priority, etc.) for the taxonomic descriptions since more complex systems typically provide multiple capabilities. For example, a specific type of air vehicle may be very agile by getting to a target very fast, in all types of weather and it can take multiple sensor readings, but it may not be agile for target location changes because it maximizes its fuel and power for the initial target. If we were to change the fuel capacity to allow for target location changes, that may increase the agility for the vehicle target changes. However, this could reduce the number of vehicle that the refueling system can support or reduce the vehicles agility because it is heavier and cannot fly as fast. Therefore, the agility of the multi-agent system may also be reduced. This is important to understand since Figure 6 shows that HMI has direct access to all three layers. When the operator wants to change the Agility, Directrability, Observability, Transparency (ADOT) of an agent within the multi-agent system, depending on where it lies, it may affect other agents. Therefore, care must be taken when making the changes, following it through the multi-agent system(s) to understand and react to the effects of the change. To illustrate this further, we will consider the changes we made to the path planning agent above.

Goal setting and Mission Monitoring	Allocation Agent	H M I
	Planning Agent	
	Execution Observations	
Mission Planning	Planning Agent	
	Task Assignment Agent	
Mission Execution	Individual Execution Agents	

Figure 6. Hierarchical Architecture of Operator-Autonomy Control System

The path planning agent sits at the lowest level of the hierarchy (see Figure 6 and Figure 7). Since it is at the bottom, it has the potential to affect the other two layers. First, we establish the perspective of each multi-agent system (i.e., hierarchy layer) based on a specific perspective in that layer when considering the system's vehicle routing capability. Then, we will analyze the ADOT for the top two layers before and after the change.

The goal setting and mission monitoring layer is there for the operator to establish mission high-level goals and monitor the mission execution. Therefore, in the perspective of this layer and the vehicle routing capability, the operator requires observability in the domain (e.g., map of the environment), the current execution status (e.g., where the vehicle is located), and the plan (e.g., the route it plans to take). Assuming that the system has vehicles that can reach anywhere in the environment, Agility is High, given targets can be set anywhere within the domain. In order for the operator to monitor the mission and set/update new goals, it would need to see the current state of the vehicles (e.g. current location of vehicles, available vehicles for future plans, planned routes, etc.). Since all of this information is made available, observability is High. In the case of directability, it is assessed in the operator's ability to display the observability features when needed. Therefore, observability is High. The operator understands that the paths generated for each vehicle are the optimally "shortest" path. How the paths are generated and what makes them the shortest is not a priority of this level and therefore the transparency of the routes can be assessed as High/Complete.

The path planning update was in response to an operator who wants to be able to direct a vehicle to avoid specific roads whether through their direction or the identification of blocked roads.

Task: Vehicle Routing						
Layer	Levels					Descriptor
	None	Minimal	Limited	Moderate	High	
Goal Setting & Mission Monitoring					Reach all Roads	Agility
					Goal Cutoff	Directability
					Routes Visible	Observability
					Shortest Path	Transparency
Mission Planning	Avoid Roads					Agility
	Avoid/Waypoints					Directability
				Distance		Observability
					Shortest Path	Transparency
Mission Execution			Avoid Roads			Agility
			Avoid Roads		Waypoints	Directability
				Distance		Observability
					Shortest Path	Transparency

Figure 7. Technique Ratings by Function

The above example illustrates how our taxonomy can be applied such that a simple agent in a well-understood domain can be improved to support HMI requirements. It also describes how feedback between the HMI and agent developer communities helped to define the desired features of the agent in a clear, concise manner, which assists in identifying needed developments. While more work is involved in implementing changes to the agent and HMI, the overall capabilities of the resulting system for future complex task environments will be enhanced by the improved coordination between the operator and autonomous agents.

4.4.2 Sense and Avoid Agent (SAA).

This example pertains to an SAA agent developed for AFRL that is designed to provide autonomous conflict avoidance capabilities for unmanned vehicles by monitoring for potential conflicts with other aircraft in the airspace and providing avoidance steering. The SAA agent's agility is High as it is highly reactive to new information obtained from sensors. It is rated Limited for Observability, since the assembled trajectories can be shown. However, these trajectories are subject to hysteresis that adversely affects the observability. Since the resulting trajectory and the sensor input are fairly straightforward to correlate, its Transparency is Moderate. Its directability, however, is Minimal, since the operator has little control over how the system responds to inputs.

AFRL determined that increases in the agent's directability would be warranted, such that the operator can improve SAA in response to changes in the operational environment. Specifically, it is desirable for the operator, via the HMI, to specify avoidance rule priorities and values (e.g., change well-clear distance), adjust the minimum separation threshold for maneuvers, and select which avoidance rules can be violated (e.g., right of way). However, the SAA agent was initially

developed to be autonomous, independent of operator input. A large part of its core functionality was built under the assumption that no HMI would be needed; and if one was needed, that the HMI would not need to interface with the agent's processing. If directability had been considered early enough, the mechanisms leveraged by the SAA algorithm could have been made adjustable. The resulting system is very limited and any changes to increase directability now (e.g., to provide access to parameters that are to be operator-adjustable) will be very difficult.

This is an example where we feel the use of our taxonomy and descriptors early in the design process to support communications between agent developers and HMI designers would have resulted in an SAA agent with more inherent directability.

4.4.3 Task Management Agent.

There has recently been increased interest not only in the mechanism by which human operators can effectively interact with autonomous systems but more specifically the ability of autonomous systems to engage in effective team-play with operators (Klein et. al., 2004). Our final example extended the initial capabilities of operator-agent interaction available via the system discussed in Section 4.4.1 to support a more robust form of autonomous task management and task allocation. Although the original system can be viewed as representing state-of-the art integration of several autonomous capabilities to support an operator's task delegation (via the operator tasking the autonomy by calling high-level "plays"), it was still limited with respect to operator-agent teaming: peer-to-peer type communication and coordination. A fundamental aspect of teamwork is allocation of the work to be performed. Such allocations typically are determined based on the capabilities and availabilities of players and the demands of the tasks to be performed. Within the original delegation scheme, the operator initiated each play in response to a mission event. There was no coordination between the operator and autonomous agents in terms of negotiating task or subtask allocation based on a teammate's capabilities or current situational demands. Even though the agent was quite capable, the original control and display interfaces did not provide adequate mechanisms to support operator-agent task coordination.

A fundamental component of the HMI dictated that the operator initiate each play in response to a mission event. The operator-driven nature of this strategy places significant demands on operators such that during high-demand situations, the operator becomes the operational bottleneck. In response, a role-driven strategy was developed that allowed for the agent to initiate those plays that were within its capacity to perform. In support of this strategy, the Task Manager panel (Figure 8) of the HMI was augmented to better support operator-agent coordination in the allocation of system tasks. Within the Task Manager, tasks were organized in rows based on priorities of the task types. The line coding used for the circle around the task icon designated if the task was assigned to be completed by the operator (solid), by the agent (dashed), or involved subtasks with both the operator and agent sharing the workload (dashed-dotted line). The Task Manager's right pane listed the subtask or steps for completing the task that was selected on the left pane. The Task Manager was designed to provide the operator the necessary insight to know what tasks were being performed and by whom, with the authority to revise the allocation at the discretion of the operator (with the constraint that tasks outside the competency boundaries of the agent could not be reassigned to the agent).

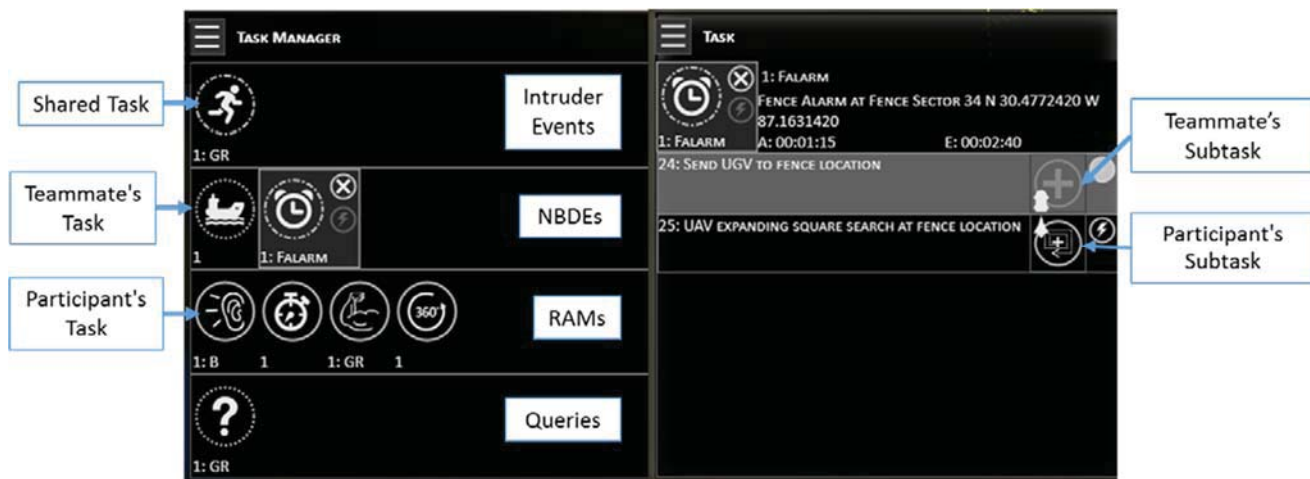


Figure 8. Task Manager for Operator and Agent Teammate Coordination

These revised interfaces were implemented to complement the other interfaces to better support operator-agent teaming and enable the evaluation of operator-driven versus role-driven team structures (Frost, et al., 2019). The interfaces informed team cognition and promoted operator-agent communication and coordination. This was accomplished by providing more status information about what tasks had been completed and what tasks needed to be completed, as well as providing information about intentions (who was responsible for completing a task, how they would complete it, and the constraints to be taken into consideration). All these enhancements increased the level of Observability of the agent's past, current, and planned actions. Moreover, revisions to the HMI, whereby the operator could influence what subtasks and tasks were performed by the agent increased the system's level of Directability.

Responses by experimental subjects participating in an assessment of the enhanced interface were generally positive. Participants reported less workload with the role-driven team structure, reflecting the additional workload associated with managing task assignment in the operator-driven structure (Frost, et al., 2018). Participants assigned to the role-driven strategy tended to rate their situational awareness higher than those assigned to the operator-driven strategy, partially attributed to the role-driven strategy having a pre-established working agreement prior to the start of missions. Participants were aware of what tasks would be automatically assigned to their agent teammate and those that would be assigned to themselves and task division was reflected in the HMI that increased Observability. Additionally, eliminating the workload associated with managing task assignment provided subjects spare capacity to monitor control station displays allowing greater attention to be allocated to situation awareness.

The refined Task Manager was reportedly useful in both team structures. Participants' ratings indicated its overall look and feel was effective as well as the interface's functionality in supporting task management (e.g., knowing the tasks assigned to each team member and whether they have been completed or not). However, their feedback suggested that the Task Manager supported coordination better for the role-driven team structure compared to the operator-driven structure in terms of understanding the teammate's actions. These ratings may also reflect that the teammate's actions were more defined in the role-driven structure.

Advancement of intelligent agent technologies are also needed to provide better autonomy transparency and support operator-agent collaboration/directability (Johnson et al., 2014). Moreover, improvements are essential to enable more dynamic task distribution and trading to help equalize workload across the team and capitalize on the capabilities of each team member. Enhancements are also required such that the agent exhibits stable performance in a range of operating conditions and mission events, notifying the human teammate when it is operating outside of its competency bounds, just as an operator would do in a team of humans.

5.0 SUMMARY

By all accounts, autonomy will play a significant role in shaping military capabilities in future conflicts. DoD planning documents suggest that autonomous systems will be the key enabler in future military systems of systems (Murphy & Shields, 2012; Masiello, 2013). A pressing question for those charged with realizing the promise of highly capable autonomous systems is delivering systems that work in the face of complex operational environments. These same planning documents highlight the need to ensure that integration with the human factor must be a central component of autonomous systems development. A great deal of research is being directed at understanding the unique demands imposed by the wide-scale adoption of autonomous systems, specifically as it relates to human-autonomy interaction. One of the key findings for current research has been the need for and significant challenges associated with the forming of a tighter coupling between those that develop autonomous agents and those that develop the interfaces to support interaction between those agents and the human operators wielding them. One of the barriers to closer coordination between these two communities is a common language and set of criteria that speaks to the unique perspectives these communities bring to the development and integration of autonomous capabilities.

Based on a comprehensive review of the relevant research we have identified and characterized a series of support functions (i.e., Agility, Directability, Observability and Transparency) and enumerated the extent to which a wide range of agent technologies effectively support these functions. Our belief is that this framework could provide a mechanism for providing human factors and autonomous systems developers a better understanding of the demands and constraints associated with the development of effective human-autonomy interaction within the context of the specific agent implementations being considered during the development process. Human factors professionals will gain greater insight into the constraints various techniques posed in support of ADOT support functions and agent developers will better understand the requirements associated with operator interfaces to autonomous systems. AFRL has applied this framework to a selection of its in-house human-autonomy development efforts and anticipates a broader application of this framework as development and integration of autonomous systems continues to expand.

6.0 REFERENCES

- Adiyatov, O. and Varol, H.A. (2013). Rapidly-exploring random tree based memory efficient motion planning. *IEEE International Conference on Mechatronics and Automation (ICMA)*, pp. 354-359.
- Bartram, L. and Ovans, R. (1995). A dialogue-based approach to the design of user interfaces for supervisory control systems. *IEEE International Conference on Systems, Man and Cybernetics, Intelligent Systems for the 21st Century*, Vol. 4, pp. 3144-3149.
- Bass, E.J., Baumgart, L.A., and Shepley, K.K. (2013). The effect of information analysis automation display content on human judgment performance in noisy environments. *Journal of Cognitive Engineering and Decision Making*, Vol.7, 1, pp.49-65.
- Burke, J.L., Murphy, R.R., Rogers, E., Lumelsky, V.J., and Scholtz, J. (2004). Final report for the DARPA/NSF interdisciplinary study on human-robot interaction. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, Vol.34, 2, pp.103-112.
- Christoffersen, K. and Woods, D.D. (2002). How to make automated systems team players. *Advances in Human Performance and Cognitive Engineering Research*, Vol.2, pp.1-12.
- Clare, A.S., Cummings, M.L., How, J.P., Whitten, A.K., and Toupet, O. (2012). Operator objective function guidance for a real-time unmanned vehicle scheduling algorithm, *Journal of Aerospace Computing, Information, and Communication*, Vol. 9, 4, pp.161-173.
- Clough, B. (2002). Relating autonomy to a task: Can it be done? *AIAA Conference on UAV Systems, Operation & Technology*, American Institute of Aeronautics and Astronautics.
- Dahm, W.J.A. (2011). Technology Horizons: A Vision for Air Force Science & Technology During 2010-2030 Volume 1, Office of the Chief Scientist of the U.S. Air Force AF/ST-TR-10-01-PR.
- DePass, B., Roth, E., Scott, R., Wampler, J., Truxler, R., and Guin, C. (2011). Designing for collaborative automation: A course of action exploration tool for transportation planning, *Proceedings of the 10th International Conference on Naturalistic Decision Making (NDM 2011)*.
- Draper, M.H. (2007). Advanced UMV operator interface. *Uninhabited Military Vehicles (UMVs): Human Factors Issues in Augmenting the Force*, NATO Report RTO-TR-HFM-078, Chapter 6. NATO RTO: Neuilly-sur-Seine, CEDEX.
- Draper, M., Rowe, A., Douglass, S., Calhoun, G., Spriggs, S., Kingston, D., and Reeder, J. (2018). Realizing autonomy via intelligent hybrid control: Adaptable autonomy for achieving UxV RSTA team decision superiority (also known as Intelligent Multi-UxV Planner with Adaptive Collaborative/Control Technologies (IMPACT)) (AFRL-RH-WP-

- TR-2018-0005). Wright-Patterson Air Force Base United States: Air Force Research Laboratory.
- Duffy, J., Hooper, D., Hughes, T. (2016). Reference document for effort examining course of action tools for decision superiority with autonomous systems (AFRL-RH-WP-TR-2016-001). Wright-Patterson Air Force Base United States: Air Force Research Laboratory.
- Frost, E.M., Bartik, J., Calhoun, G., Spriggs, S., Ruff, H., and Behymer, K. (2018). Creating a well-situation human-autonomy team: The effects of team structure. *Proceedings of the NATO Human Factors and Medicine Panel, Symposium on Human Autonomy Teaming*, STO-MP-HFM-300.
- Frost, E., Calhoun, G., Ruff, H., Bartik, J., Behymer, K., Spriggs, S., and Buchanan, A. (2019). Collaboration interface supporting human-autonomy teaming for unmanned vehicle management. *International Symposium of Aviation Psychology*.
- Goodrich, M.A. and Olsen, D.R. (2003). Seven principles of efficient human robot interaction. *IEEE International Conference on Systems, Man and Cybernetics*, Vol. 4, pp. 3942-3948.
- Hansen, M., Calhoun, G., Douglass, S. A., & Evans, D. (2016). Courses of action display for multi-unmanned vehicle control: A multi-disciplinary approach. *2016 AAAI Fall Symposium Series*.
- Hooper, D.J., Duffy, J.P., Calhoun, G.L., and Hughes, T.L. (2015). A taxonomy for improving dialog between autonomous agent developers and human machine interface designers. *Artificial Intelligence and Human-Robot Interaction, AAAI Fall Symposium Series*.
- Johnson, M., Bradshaw, J.M., Hoffman, R.R., Feltovich, P.J., and Woods, D.D. (2014). Seven cardinal virtues of human-machine teamwork: Examples from the DARPA robotic challenge. *IEEE Intelligent Systems*, 29(6), 74-80.
- Klein, G., Woods, D.D., Bradshaw, J.M., Hoffman, R.R., and Feltovich, P.J., (2004), Ten challenges for making automation a "team player" in joint human-agent activity. *IEEE Intelligent Systems*, 6, pp.91-95.
- Korsah, G.A., Stentz, A., and Dias, M.B. (2013). A comprehensive taxonomy for multi-robot task allocation. *The International Journal of Robotics Research*, Vol. 32, 12, pp.1495-1512.
- Masiello, T. (2013). Air Force Research Laboratory Autonomy Science and Technology Strategy, Air Force Research Laboratory, Wright-Patterson Air Force Base, OH.
- Morley, K.L. and Myers, D.N. (2001). Human directability of agents. *Proceedings of the International Conference on Knowledge Capture*, ACM, pp. 108-115.
- Murphy, R. and Shields, J. (2012). Department of Defense - Defense Science Board Task Force Report: The Role of Autonomy in DoD Systems, Office of The Under Secretary of Defense for Acquisitions, Technology and Logistics.

- Olson, W.A. and Wuennenberg, M.G. (2001). Autonomy based human-vehicle interface standards for remotely operated aircraft. *Digital Avionics Systems (DASC.) 20th Conference*, Vol. 2, pp. 7D3/1-7D3/9.
- Russell, S. and Norvig, P. (2003). *Artificial Intelligence: A Modern Approach* (2nd Edition), Prentice Hall.
- Steinfeld, A. (2004). Interface lessons for fully and semi-autonomous mobile robots. *Proceedings IEEE International Conference on Robotics and Automation, ICRA*, Vol. 3, pp. 2752-2757.
- Truxler, R., Roth, E., Scott, R., Smith, S. and Wampler, J. (2012). Designing collaborative automated planners for agile adaptation to dynamic change. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, SAGE Publications, Vol. 56, pp. 223-227.
- Woods, D.D and Hollnagel, E. (2006). *Joint Cognitive Systems: Patterns in Cognitive Systems Engineering*, CRC Press

Appendix A - Bibliography

- Aberdeen, D., Thiebaut, S., and Zhang, Z. (2004). Decision-theoretic military operations planning. *ICAPS*, pp. 402-412.
- Abreu, N. and Matos, A. (2014). Using evolutionary algorithms to plan automatic mine hunting operations. *11th International Conference on Informatics in Control, Automation and Robotics (ICINCO)*, Vol. 1, pp. 228-235.
- Adiyatov, O. and Varol, H. A. (2013). Rapidly-exploring random tree based memory efficient motion planning. *IEEE International Conference on Mechatronics and Automation (ICMA)*, pp. 354-359.
- Afsar, S., Malik, Z.I., and Elnaffar, S. (2010). Autonomous agent oriented traffic control system. *The 2nd International Conference on Computer and Automation Engineering (ICCAE)*, Vol. 2, pp. 321-324.
- Alcazar, V., Veloso, M., and Borrajo, D. (2011). Adapting a rapidly-exploring random tree for automated planning. *The Fourth International Symposium on Combinatorial Search (S0CS-2011)*, Castell de Cardona, Barcelona, Spain.
- Allamaraju, R., H. Kingravi, H., Axelrod, A., Chowdhary, G., Grande, R., How, J.P., Crick, C., and Sheng W. (2014). Human aware UAS path planning in urban environments using nonstationary MDPs. *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1161-1167.
- Arslan, O. and Tsiotras, P. (2013). Use of relaxation methods in sampling-based algorithms for optimal motion planning. *IEEE International Conference on Robotics and Automation (ICRA 2013)*, Karlsruhe, Germany, pp. 2421.
- Bartram, L. and Ovans, R. (1995). A dialogue-based approach to the design of user interfaces for supervisory control systems. *IEEE International Conference on Systems, Man and Cybernetics, Intelligent Systems for the 21st Century*, Vol. 4, pp. 3144-3149.
- Bass, E.J., Baumgart, L.A., and Shepley, K.K. (2013). The effect of information analysis automation display content on human judgment performance in noisy environments. *Journal of Cognitive Engineering and Decision Making*, Vol.7, 1, pp.49-65.
- Basso, B., Durrant-Whyte, H., and Hedrick, H.A. (2011). Mission planning in unstructured environments: A reinforcement learning approach. *50th IEEE Conference on Decision and Control and European Control Conference (CDC-ECC)*, pp. 578-583.
- Beamer, S., Asanovic, K., and Patterson, D. (2012). Direction-optimizing breadth-first search. *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pp. 1-10.
- Bindewald, J.M. (2014). The effect of similarity between human and machine action choices on adaptive automation performance. *Twenty-Eighth AAAI Conference on Artificial Intelligence*.

- Bindewald, J.M., Miller, M.E., and Peterson, G.L. (2014). A function-to-task process model for adaptive automation system design. *International Journal of Human-Computer Studies*, Vol.72, 12, pp.822-834.
- Bindewald, J.M., Peterson, G.L., and Miller, M.E. (2015). Trajectory generation with player modeling. *Advances in Artificial Intelligence*, Springer, pp.42-49.
- Bryant, C.H. (2006). Robust planning for effects-based operations, Masters of Science Thesis, Massachusetts Institute of Technology, Sloan School of Management.
- Burke, J.L., Murphy, R.R., Rogers, E., Lumelsky, V.J., and Scholtz, J. (2004). Final report for the DARPA/NSF interdisciplinary study on human-robot interaction. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, Vol. 34, 2, pp.103-112.
- Chen, J.Y.C., Barnes, M.J., and Harper-Sciarini, M. (2011). Supervisory control of multiple robots: Human-performance issues and user-interface design. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, Vol.41, 4, pp.435-454.
- Christoffersen, K. and Woods, D.D. (2002). How to make automated systems team players, *Advances in Human Performance and Cognitive Engineering Research*, Vol.2, pp.1-12.
- Clare, A.S., Cummings, M.L., How, J.P., Whitten, A.K., and Toupet, O. (2012). Operator objective function guidance for a real-time unmanned vehicle scheduling algorithm. *Journal of Aerospace Computing, Information, and Communication*, Vol.9, 4, pp.161-173.
- Clough, B. (2002). Relating autonomy to a task: Can it be done? *AIAA Conference on UAV Systems, Operation & Technology*, American Institute of Aeronautics and Astronautics.
- Crescenzi, T., Kaizer, A., Young, T., Holt, J. and Biaz, S. (2011). Collision avoidance in UAVs using dynamic sparse A*, Technical Report #CSSE11 - 02, Auburn University.
- Cully, A., Clune, J., Tarapore, D., and Mouret, J. (2015). Robots that can adapt like animals. *Nature*, Vol.521, 7553, pp.503-507.
- Cummings, M.L. and Guerlain, S. (2007). Developing operator capacity estimates for supervisory control of autonomous vehicles, *Human Factors*, Vol.49, 1, pp.1-15.
- Dahm, W.J.A. (2011). Technology Horizons: A Vision for Air Force Science & Technology During 2010-2030 Volume 1, Office of the Chief Scientist of the U.S. Air Force AF/ST-TR-10-01-PR.
- DePass, B., Roth, E., Scott, R., Wampler, J., Truxler, R. and Guin, C. (2011). Designing for collaborative automation: A course of action exploration tool for transportation planning.

Proceedings of the 10th International Conference on Naturalistic Decision Making (NDM 2011).

- Desai, M., Medvedev, M., Vazquez, M., McSheehy, S., Gadea-Omelchenko, S., Bruggeman, C., Steinfeld, A., and Yanco, H. (2012). Effects of changing reliability on trust of robot systems. *7th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pp. 73-80, isbn=2167-2121
- Dias, M.B., Koenig, S., Lagoudakis, M.G., Zlot, R., Kalra, N., and Jones, G. (2006). Tutorial on auction-based agent coordination. *The Twenty-first National Conference on Artificial Intelligence (AAAI-06) Tutorial Forum*, Boston, MA.
- Dias, M.B. and Stentz, A. (2000). A free market architecture for distributed control of a multirobot system. *6th International Conference on Intelligent Autonomous Systems (IAS-6)*, pp. 115-122.
- Dixon, C. (1996). Temporal resolution: a breadth-first search approach. *Proceedings of Third International Workshop on Temporal Representation and Reasoning, (TIME '96)*, pp. 120-127.
- Drury, J.L., Scholtz, J., and Yanco, H.A. (2006). Applying CSCW and HCI Techniques to Human-Robot Interaction, MITRE Corporation, Bedford MA, DTIC ADA 456 188.
- Dudek, G., Jenkin, M.M., Milios, E., and Wilkes, D. (1996). A taxonomy for multi-agent robotics, *Autonomous Robots*, Vol.3, 4, pp. 375-397.
- Duffy, J.P. (2008), Dynamic behavior sequencing in a hybrid robot architecture, Masters of Science Thesis, Air Force Institute of Technology (No. AFIT/GCE/ENG/08-03).
- Duffy, J.P. and Peterson, G.L. (2008). An Abstract Behavior Representation for Robust, Dynamic Sequencing in a Hybrid Architecture. *AAAI Spring Symposium: Emotion, Personality, and Social Behavior*, pp. 19-24.
- Duhamel, C., Lacomme, P., and Prodhon, C. (2011). Efficient frameworks for greedy split and new depth first search split procedures for routing problems. *Computers & Operations Research*, Vol.38, 4, pp.723.
- Duhamel, C., Lacomme, P., and Prodhon, C. (2012). A hybrid evolutionary local search with depth first search split procedure for the heterogeneous vehicle routing problems. *Engineering Applications of Artificial Intelligence*, Vol.25, 2, pp.345-358.
- Duran, B. and Sandamirskaya, Y. (2012). Neural dynamics of hierarchically organized sequences: A robotic implementation. *12th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pp. 357-362.
- DuToit, R., Lyle, M., Holt, M., and Biaz, S. (2012). UAV collision avoidance using RRT* and LOS Maximization, Technical Report #CSSE12 - 03, Auburn University.

- Farinelli, A., Iocchi, L. and Nardi, D. (2004). Multirobot systems: A classification focused on coordination. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, Vol. 34, 5, pp.2015-2028.
- Ferguson, D. and Stentz, A. (2004). Focused processing of MDPs for path planning. *16th IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pp. 310-317.
- Firestone, S.J. (2002). A partially observable approach to allocating resources in a dynamic battle scenario. Doctoral dissertation, Massachusetts Institute of Technology.
- Franklin, S. and Graesser, A. (1997). Is it an agent, or just a program?: A taxonomy for autonomous agents, *International Workshop on Agent Theories, Architectures, and Languages*. Springer, Vol.1193, pp.21-35.
- Gammell, J.D., Srinivasa, S.S., and Barfoot, T.D. (2014). Informed RRT*: Optimal incremental path planning focused through an admissible ellipsoidal heuristic. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Vol. 2(1), pp. 3-1.
- Gammell, J.D., Srinivasa, S.S., and Barfoot, T.D. (2015). Batch Informed Trees (BIT*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs. *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3067-3074.
- Gao, M. and Tian, J. (2007). Planning - path for mobile robot based on improved simulated annealing artificial neural network. *Third International Conference on Natural Computation (ICNC)*, Vol. 3, pp. 8-12.
- Gargiulo, F. and Quagliarella, D. (2012). Genetic algorithms for the resource constrained project scheduling problem. *IEEE 13th International Symposium on Computational Intelligence and Informatics (CINTI)*, pp. 39-47.
- Gautam, S.A. and Verma, N. (2014). Path planning for unmanned aerial vehicle based on genetic algorithm & artificial neural network in 3D. *IEEE International Conference on Data Mining and Intelligent Computing (ICDMIC)*, pp. 1-5.
- Gerkey, B.P. and Mataric, M.J. (2002). Sold!: Auction methods for multirobot coordination. *IEEE Transactions on Robotics and Automation*, Vol.18, 5, pp.758-768.
- Gerkey, B.P. and Mataric, M.J. (2003). Multi-robot task allocation: Analyzing the complexity and optimality of key architectures. *Proceedings of the IEEE Intl. Conf. on Robotics and Automation (ICRA)*, Taipei, Taiwan, pp. 3862-3868.
- Gerkey, B.P. and Mataric, M.J. (2004). A formal analysis and taxonomy of task allocation in multi-robot systems. *The International Journal of Robotics Research*, Vol. 23, 9, pp.939-954.

- Gmytrasiewicz, P.J. and Doshi, P. (2005). A framework for sequential planning in multi-agent settings. *Journal of Artificial Intelligence Research (JAIR)*, Vol.24, pp.49-79.
- Gombolay, M., Wilcox, R. and Shah, J.A. (2013). Fast scheduling of multi-robot teams with temporospatial constraints. *Robotics: Science and Systems*. Available at DSpace@MIT Home.
- Goodrich, M.A. (2002). Potential Fields Tutorial (http://phoenix.goucher.edu/~jillz/cs325_robotics/goodrich_potential_fields.pdf).
- Goodrich, M.A. and Olsen, D.R. (2003). Seven principles of efficient human robot interaction. *IEEE International Conference on Systems, Man and Cybernetics*, Vol. 4, pp. 3942-3948.
- Hanna, H. (2005). Decentralized approach for multi-robot task allocation problem with uncertain task execution. *IEEE/RSJ International Conference on Intelligent Robots and Systems, (IROS 2005)*, pp. 535-540.
- Hao, Y., Shen, Z., and Zhao, Y. (2009). –Path planning for aircraft based on MAKLINK Graph Theory and Multi Colony Ant Algorithm. *International Joint Conference on Computational Sciences and Optimization (CSO 2009)*, Vol. 2, pp. 232-235.
- Haoyu Bai, H., Hsu, D., Kochenderfer, M.J., and Sun Lee, W. (2012). Unmanned aircraft collision avoidance using continuous-state POMDPs. *Robotics: Science and Systems*, Vol. 7, pp. 1.
- Heo, Y.J. and Chung, W.C. (2013). RRT-based path planning with kinematic constraints of AUV in underwater structured environment. *10th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI 2013)*, pp. 523-525.
- Holt, J., Biaz, S., Yilmaz, L., and Aji, C.A. (2014). Symbiotic simulation architecture for evaluating UAVs. *Journal of Simulation*, Vol. 8, 1, pp.64-75.
- Hooper, D. (2008). Tightly coupled cooperation among independent agents. *Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 3, AAAI'08*, AAAI Press, Chicago, Illinois, pp. 1853-1854.
- Hooper, D. (2010). Coalition formation under uncertainty. Doctoral Dissertation, Air Force Institute of Technology (No. AFIT/DEE/ENG/10-05).
- Humphrey, C.H., Gordon, S.M., and Adams, J.A. (2006). Visualization of multiple robots during team activities. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, SAGE Publications, Vol. 50, pp. 651-655.
- Ibekwe, H.I. and Kamrani, A.K. (2014). Navigation for autonomous robots in partially observable facilities. *World Automation Congress (WAC), 2014*, pp. 1-5.

- Islam, F., Nasir, J., Malik, U., Ayaz, Y., and Hasan, O. (2012). RRT*-Smart: Rapid convergence implementation of RRT* towards optimal solution. *International Conference on Mechatronics and Automation (ICMA 2012)*, pp. 1651-1656.
- Jacob, R. and Hedrick, J.K. (2014). directPath: An expert path planning framework to handle environment knowledge. *IEEE 4th Annual International Conference on Cyber Technology in Automation, Control, and Intelligent Systems (CYBER 2014)*, pp. 7-12.
- Janabi-Sharifi, F. and Vinke, D. (1993). Integration of the artificial potential field approach with simulated annealing for robot path planning. *Proceedings of the 1993 IEEE International Symposium on Intelligent Control*, pp. 536-541.
- Jezequel, L. and Fabre, E. (2012). A#: A distributed version of A* for factored planning. *IEEE 51st Annual Conference on Decision and Control (CDC 2012)*, pp. 7377-7382.
- Johnson, M., Bradshaw, J.M., Feltovich, P.J., Hoffman, R.R., Jonker, C., van Riemsdijk, B., and Sierhuis, M. (2011). Beyond cooperative robotics: The central role of interdependence in coactive design, intelligent systems. *IEEE Intelligent Systems*, Vol. 26, 3, pp.81-88.
- Kalman, R. (1959). On the general theory of control systems. *IRE Transactions on Automatic Control*, Vol. 4, 3, pp.110-110.
- Karaman, S., & Frazzoli, E. (2011). Sampling-based algorithms for optimal motion planning. *The international journal of robotics research*, 30(7), 846-894.
- Katasonov, A. and Terziyan, V. (2009). Semantic approach to dynamic coordination in autonomous systems. *IEEE Fifth International Conference on Autonomic and Autonomous Systems*, pp. 321-329.
- Katsavounis, S. (2006). Scheduling tasks with execution and precedence constraints using longest compatible paths. *Information and Communication Technologies, ICTTA*, Vol. 2, pp. 3474-3479.
- Kawano, H. (2005). Method for applying reinforcement learning to motion planning and control of under-actuated underwater vehicle in unknown non-uniform sea flow. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 996-1002.
- Keller, I., Visser, F.S., van der Lugt, R., and Stappers, P.J. (2009). Collecting with cabinet: or how designers organise visual material, researched through an experiential prototype. *Design Studies*, Vol.30, 1, pp.69-86.
- Khamis, A.M., Elmogy, A.M., and Karray, F.O. (2011). Complex task allocation in mobile surveillance systems. *Journal of Intelligent & Robotic Systems*, Vol. 64, 1, pp.33-55.
- Kim, J. and Esposito, J.M. (2005). Adaptive sample bias for rapidly-exploring random trees with applications to test generation. *Proceedings of the 2005 American Control Conference*, Vol. 2, pp. 1166-1172.

- Klein, D. and Abbeel, P. (2015). UC Berkeley CS188 Intro to AI -- Course Materials (<http://ai.berkeley.edu/>).
- Klein, G., Woods, D.D., Bradshaw, J.M., Hoffman, R.R., and Feltovich, P.J. (2004). Ten challenges for making automation a "team player" in joint human-agent activity. *IEEE Intelligent Systems*, 6, pp.91-95.
- Kochenderfer, M.J. and Chryssanthacopoulos, J.P. (2010). A decision-theoretic approach to developing robust collision avoidance logic. *13th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pp. 1837-1842.
- Kohl, N. and Stone, P. (2004). Policy gradient reinforcement learning for fast quadrupedal locomotion. *IEEE International Conference on Robotics and Automation (ICRA)*, Vol. 3, pp. 2619-2624.
- Korsah, G.A., Stentz, A., and Dias, M.B. (2013). A comprehensive taxonomy for multi-robot task allocation. *The International Journal of Robotics Research*, Vol.32, 12, pp.1495-1512.
- Kuffner, J.J. and LaValle, S.M. (2000). RRT-connect: An efficient approach to single-query path planning, *IEEE International Conference on Robotics and Automation (ICRA)*, Vol. 2, pp. 995-1001.
- Kuhlman, M.J., Svec, P., Kaipa, K.N., Sofge, D., and Gupta, S.K. (2014). Physics-aware informative coverage planning for autonomous vehicles. *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4741-4746.
- Kuwata, Y., Karaman, S., Teo, J., Frazzoli, E., How, J.P., and Fiore, G. (2009). Real-time motion planning with applications to autonomous urban driving. *IEEE Transactions on Control Systems Technology*, Vol.17, 5, pp.1105-1118.
- Lange, D.S., Verbancsics, P., Gutzwiller, R., and Reeder, J. (2012). Command and control of teams of autonomous units. *17th International Command & Control Research & Technology Symposium (ICCRTS)*.
- Laroche, P. (2000). Building efficient partial plans using Markov decision processes. *12th IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pp. 156-163.
- LaValle, S.M. (1998). Rapidly Exploring Random Trees: A New Tool for Planning - Path, Technical Report 98-11, Iowa State University.
- Lei Gang, L., Min-zhou, D., Tao, X., and Liang, W. (2011). Multi-agent planning - path for unmanned aerial vehicle based on threats analysis. *3rd International Workshop on Intelligent Systems and Applications (ISA)*, pp. 1-4.
- Leonard, J.J. and Durrant-Whyte, H.F. (1991). Mobile robot localization by tracking geometric beacons. *IEEE Transactions on Robotics and Automation*, Vol.7, 3, pp.376-382.

- Leonard, J.J. and Durrant-Whyte, H.F. (1991). Simultaneous map building and localization for an autonomous mobile robot. *IEEE International Workshop on Intelligent Robots and Systems (IROS)*, pp. 1442-1447.
- Levine, D.S. (2014). Neurocognition, controls, efficient communication, and enhanced decision for fast satisficing in autonomous military systems. *Office of Naval Research Science of Autonomy Meeting*, Arlington, VA
- Li, B., Chang, J., and Wu, C. (2012). A potential function and artificial neural network for path planning in dynamic environments based on self-reconfigurable mobile robot system. *IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, pp. 1-6.
- Lin, L. and Goodrich, M.A. (2014). Hierarchical heuristic search using a Gaussian Mixture Model for UAV coverage planning. *IEEE Transactions on Cybernetics*, Vol. 44, 12, pp. 2532-2544.
- Lin, Y. and Saripalli, S. (2014). Path planning using 3D Dubins Curve for unmanned aerial vehicles. *International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 296-304.
- Liu, Y., Pan, Z., Stirling, D., and Naghdy, F. (2009). Control of autonomous airship. *IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pp. 2457-2462.
- Lu, S., Paris, C., and Vander Linden, K. (1999). Toward the automatic construction of task models from object-oriented diagrams. *IFIP International Conference on Engineering for Human-Computer Interaction*, Vol. 22, pp. 169-189, Springer.
- Ma, G., Duan, H., and Liu, S. (2007). Improved Ant Colony Algorithm for global optimal trajectory planning of UAV under complex environment. *International Journal of Computer Science & Applications*. Vol. 4, 3, pp. 57-68.
- Marquardt, D.W. (1963) .An algorithm for least-squares estimation of nonlinear parameters. *Journal of the Society for Industrial & Applied Mathematics*, Vol. 11, 2, pp. 431-441.
- Masiello, T. (2013). Air Force Research Laboratory Autonomy Science and Technology Strategy, Air Force Research Laboratory, Wright-Patterson AFB, OH.
- Merkle, D., Middendorf, M., and Schmeck, H. (2002). Ant colony optimization for resource-constrained project scheduling. *IEEE Transactions on Evolutionary Computation*, Vol.6, 4, pp. 333-346.
- Mettler, B., Feit, A., Verma, A., and Li, B. (2014). Modeling human cognitive functions in vehicle guidance from invariant features in agent-environment interactions. *Office of Naval Research Science of Autonomy Meeting*, Arlington, VA.
- Meuleau, N., Hauskrecht, M., Kim, K.E., Peshkin, L., Kaelbling, L.P., Dean, T., and Boutilier, C. (1998). Solving very large weakly coupled Markov Decision Processes. *Proceedings of the Fifteenth National/Tenth Conference on Artificial Intelligence/Innovative*

- Applications of Artificial Intelligence*, (IAAI), American Association for Artificial Intelligence, Menlo Park, CA, USA, Madison, WI, pp. 165-172.
- Miao, H. and Tian, Y. (2008). Robot path planning in dynamic environments using a simulated annealing based approach. *10th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, pp. 1253-1258.
- Mitchell, T.M. (1997). *Machine Learning*, MacGraw-Hill, Vol.1, pp.997.
- Murphy, R. and Shields, J. (2012). Department of Defense - Defense Science Board Task Force Report: The Role of Autonomy in DoD Systems, Office of The Under Secretary of Defense for Acquisitions, Technology and Logistics.
- Myers, D.N. and Morley, K.L. (2001). Human directability of agents. *Proceedings of the First International Conference on Knowledge Capture*, ACM, pp. 108-115.
- Narayanan, V., Zhang, Y., Mendoza, N., and Kambhampati, S. (2014). Plan or not: Remote human-robot teaming with incomplete task information, arXiv preprint arXiv: 1412.2824
- Navarre, D., Palanque, P., Bastide, R., and Su, O. (2001). A model-based tool for interactive prototyping of highly interactive applications. *12th International Workshop on Rapid System Prototyping*, pp. 136-141.
- Olson, W.A. and Wuennenberg, M.G. (2001). Autonomy based human-vehicle interface standards for remotely operated aircraft. *20th Digital Avionics Systems Conference (DASC)*, Vol. 2, pp. 7D3/1-7D3/9.
- Paduano, J. (2014). Application: Autonomous aerial cargo/utility system (AACUS). *ONR Science of Autonomy Meeting*, Arlington, VA.
- Park, M.G. and Lee, M.C. (2002). Experimental evaluation of robot path planning by artificial potential field approach with simulated annealing. *Proceedings of the 41st SICE Annual Conference*, Vol. 4, pp. 2190-2195.
- Parunak, H. Van Dyke and Brueckner, S.A. (2003). Dynamic imputation of agent cognition. *International Workshop on Computational Autonomy*, Vol. 2969, pp. 209-226.
- Parunak, H. Van Dyke, Brueckner, S., Fleischer, M., and Odell, J. (2004). A design taxonomy of multi-agent interactions, *International Workshop on Agent-Oriented Software Engineering*, Vol. 2935, pp. 123-137.
- Paterno, F., Mancini, C., and Meniconi, S. (1997). ConcurTaskTrees: A diagrammatic notation for specifying task models. *Human-computer Interaction INTERACT*, Springer, pp. 362-369.
- Perez, A., Platt, R., Konidaris, G., Kaelbling, L., and Lozano-Perez, T. (2012). LQR-RRT*: Optimal sampling-based motion planning with automatically derived extension

- heuristics, *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2537-2542.
- Peterson, G.L., Duffy, J.P., and Hooper, D.J. (2011). Dynamic behavior sequencing for hybrid robot architectures. *Journal of Intelligent & Robotic Systems*, Vol. 64, 2, pp.179-196.
- Pohl, A. J. and Lamont, G.B. (2008). Multi-objective UAV mission planning using evolutionary computation., *Winter Simulation Conference (WSC)*, pp. 1268-1279.
- Poole, D.L. and Mackworth, A.K. (2010). *Artificial intelligence: Foundations of computational agents*. Cambridge University Press, New York, NY.
- Port, R.F. and Van Gelder, T. (1995). *Mind as motion: Explorations in the dynamics of cognition*, Massachusetts Institute of Technology Press, pp.1-43.
- Ramchurn, S.D., Huynh, T.D., Ikuno, Y., Flann, J., Wu, F., Moreau, L., Jennings, N.R., Fischer, J.E., Jiang, W., Rodden, T., Simpson, E., Reece, S., and Roberts, S.J. (2015). HAC-ER: A disaster response system based on human-agent collectives. *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. International Foundation for Autonomous Agents and Multiagent Systems, Istanbul, Turkey, pp. 533-541.
- Rasche, C., Stern, C., Kleinjohann, L., and Kleinjohann, B. (2010). Role-based path planning and task allocation with exploration tradeoff for UAVs. *11th International Conference on Control Automation Robotics & Vision (ICARCV)*, pp. 417-422.
- Rasmussen, S., Kingston, D., & Humphrey, L. (2018, June). A brief introduction to unmanned systems autonomy services (UxAS). *2018 International Conference on Unmanned Aircraft Systems (ICUAS)*, 257-268.
- Rimal, B.P., Shin, H., and Choi, E., (2009). Simulation of nonlinear identification and control of unmanned aerial vehicle: An artificial neural network approach. *9th International Symposium on Communications and Information Technology (ISCIT)*, pp. 442-447.
- Russell, S. and Norvig, P. (2003). *Artificial intelligence: A modern approach* (2nd Edition), Prentice Hall.
- Sahingoz, O.K. (2013). Flyable path planning for a multi-UAV system with Genetic Algorithms and Bezier curves. *International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 41-48.
- Sankar, K. and Krishnamoorthy, K. (2010). Ant Colony algorithm for routing problem using rule-mining. *IEEE International Conference on Computational Intelligence and Computing Research (ICCIC)*, pp. 1-8.
- Scholtz, J. (2003). Theory and evaluation of human robot interactions. *Proceedings of the 36th Annual Hawaii International Conference on System Sciences*, pp. 10.

- Sen, S.D. and Adams, J.A. (2014). An influence diagram based multi-criteria decision making framework for multirobot coalition formation. *Autonomous Agents and Multi-Agent Systems*, pp.1-30.
- Seneler, C.O., Basoglu, N., and Daim, T.U. (2008). A taxonomy for technology adoption: A human computer interaction perspective. *Portland International Conference on Management of Engineering & Technology (PICMET)*, pp. 2208-2219.
- Shiffman, D., Fry, S., and Marsh, Z. (2012). *The Nature of Code*, pp. 323-330.
- Silvela, J. and Portillo, J. (2001). Breadth-first search and its application to image processing problems. *IEEE Transactions on Image Processing*, Vol. 10, 8, pp. 1194-1199.
- Steinfeld, A. (2004). Interface lessons for fully and semi-autonomous mobile robots. *IEEE International Conference on Robotics and Automation (ICRA)*, Vol. 3, pp. 2752-2757.
- Steinfeld, A., Fong, T., Kaber, D., Lewis, M., Scholtz, J., Schultz, A., and Goodrich, M. (2006). Common metrics for human-robot interaction. *Proceedings of the 1st ACM SIGCHI/SIGART Conference on Human-Robot Interaction*, ACM, pp. 33-40.
- Szczerba, R.J., Galkowski, P., Glicktein, I.S., and Ternullo, N. (2000). Robust algorithm for real-time route planning. *IEEE Transactions on Aerospace and Electronic Systems*, Vol. 36, 3, pp.869-878.
- Taha, T., Miro, J.V., and Dissanayake, G. (2008). POMDP-based long-term user intention prediction for wheelchair navigation. *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3920-3925.
- Tarby, J.C. and Barthet, M.F. (1996). The DIANE+ Method. *Computer-Aided Design of User Interface (CADUI)*, Vol. 96, pp. 95-119.
- Temizer, S., Kochenderfer, M.J., Kaelbling, L.P., Lozano-Perez, T., and Kuchar, J.K. (2010). Collision avoidance for unmanned aircraft using Markov decision processes. *AIAA Guidance, Navigation, and Control Conference*.
- Truxler, R., Roth, E., Scott, R., Smith, S., and Wampler, J. (2012). Designing collaborative automated planners for agile adaptation to dynamic change. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, SAGE Publications, Vol. 56, pp. 223-227.
- Tsiotras, P., Soatto, S., Itti, L., and Schaal, S. (2014). Bounded rationality autonomy using neuromorphic decision and action models: Proper use of resources. *Office of Naval Research Science of Autonomy Meeting*, Arlington, VA.
- Tuna, G., Gungor, V.C., and Gulez, K. (2014). An autonomous wireless sensor network deployment system using mobile robots for human existence detection in case of disasters. *Ad Hoc Networks*, Vol. 13, Part A, pp. 54-68.

- Uny Cao, Y., Fukunaga, A., and Kahng, A. (1997). Cooperative mobile robotics: Antecedents and directions. *Autonomous Robots*, Vol. 4, 1, pp. 7-27.
- Vassev, E. and Hinchey, M. (2013). Autonomy requirements engineering. *IEEE 14th International Conference on Information Reuse and Integration (IRI)*, pp. 175-184.
- Wang, Y. (2009). A Cognitive informatics reference model of autonomous agent systems (AAS), *International Journal of Cognitive Informatics and Natural Intelligence (IJCINI)*, Vol. 3, 1, pp. 1-16,
- Wang, Z., Liu, L., Long, T., Yu, C., and Kou, J. (2014). Enhanced sparse A* search for UAV path planning using dubins path estimation. *33rd Chinese Control Conference (CCC)*, pp. 738-742.
- Weld, D.S. (2004). Solving concurrent Markov Decision Processes. *Proceedings of the 19th National Conference on Artificial Intelligence*, AAAI Press, San Jose, California, pp. 716-722.
- Westing, N.M. (2014). Carbon nanotube growth rate regression using support vector machines and artificial neural networks. Master of Science Thesis, Air Force Institute of Technology.
- Woods, D.D. and Hollnagel, E. (2006). Joint cognitive systems: Patterns in cognitive systems engineering, CRC Press.
- Wu, B., Ke, Q., and Dong, Y. (2011). Degree and similarity based search in networks. *Eighth International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*, Vol. 2, pp. 1267-1270.
- Yanco, H.A. and Drury, J.L. (2002). A taxonomy for human-robot interaction. *Proceedings of the AAAI Fall Symposium on Human-Robot Interaction*, pp. 111-119,
- Yanco, H.A. and Drury, J.L. (2004). Classifying human-robot interaction: an updated taxonomy. *IEEE International Conference on Systems, Man and Cybernetics*, Vol. 3, pp. 2841-2846.
- Yershova, A., Jaillet, L., Simeon, T., and LaValle, S.M. (2005). Dynamic-domain RRTs: Efficient exploration by controlling the sampling domain. *Proceedings of the 2005 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3856-3861.
- Yosinski, J., Clune, J., Bengio, Y., and Lipson, H. (2014). How transferable are features in deep neural networks. *Advances in Neural Information Processing Systems*, pp. 3320-3328.
- Yun, L. (2002). The application of breadth first search in the dispatching system of the China Railway Communication network. *IEEE Region 10 Conference on Computers, Communications, Control and Power Engineering (TENCON)*, Vol. 2, pp. 863-868.

- Zaza, T. and Richards, A. (2014). Ant Colony Optimization for routing and tasking problems for teams of UAVs. *International Conference on Control (CONTROL), UKACC*, pp. 652-655.
- Zhang, Q., Ma, J., and Liu, Q. (2012). Path planning based quadtree representation for mobile robot using hybrid-simulated annealing and ant colony optimization algorithm. *10th World Congress on Intelligent Control and Automation (WCICA)*, pp. 2537-2542.
- Zhang, R., Zheng, C., and Yan, P. (2007). Route planning for unmanned air vehicles with multiple missions using an evolutionary algorithm. *Third International Conference on Natural Computation (ICNC)*, Vol. 3, pp. 23-28.
- Zhu, Q., Yan, Y. and Xing, Z., (2006). Robot path planning based on artificial potential field approach with simulated annealing. *Sixth International Conference on Intelligent Systems Design and Applications, (ISDA)*, Vol. 2, pp. 622-627.

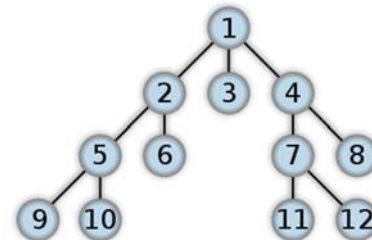
Appendix B - Tutorials

Breadth-First Search (BFS)

The BFS is an uninformed search that is performed on a tree, a non-cyclical graph, or a graph where a re-visit of the same state will have the same potential ensuing states. The BFS is a widely applicable search that can be used whenever there are sufficient computing resources and the goal is to find a solution with the lowest unit step cost. The BFS is a simple, straightforward, parallelizable method for evaluating the shortest path to a goal.

In order for the BFS to work as designed, the following assumptions must be true:

1. States are fully defined (i.e., it is easy to tell whether the agent has entered a given state)
2. Actions are deterministic
3. Re-visits of the same state will have the same potential ensuing states
4. World state does not change during deliberation



BFS selects the shallowest unexpanded node in the search tree for expansion. In general, all nodes at a given depth in the tree are expanded before any other nodes are expanded at the next level. The figure above shows the order in which the BFS expands the nodes. As shown, the nodes of the deeper levels are not explored until all nodes of the shallower levels are explored. This makes the BFS very slow for solutions that are at extreme depths within the tree. This search is complete and optimal. It is optimal in that the first solution it finds will have the lowest unit step cost. In other words, it will find the solution that requires the least amount of actions to reach the goal. BFS is complete because it will find a solution if one exists.

BFS is not agile. This search cannot react to changes related to the search tree caused by relevant environment changes. If there is a change in the search tree, the algorithm must start over since the previous tree no longer represents the current environment.

There is no directability in BFS. The algorithm runs as expected and will return the same solution, on the same search tree, every time. If the ordering of action selection is changed, a new solution may be found *only* if it is at the same depth as the alternative one.

The BFS algorithm accesses or holds all the relevant information to construct the complete state of the search at any time. If the accessed states are made available, then the searched tree is observable. If the pending list is made available, then the current state of the search is observable. Therefore, the BFS is completely observable if the accessed states and pending states are made available. Otherwise, it is minimally observable if either the accessed states or the pending states are made available.

Since BFS is a deterministic algorithm that will always return the same solution on the same search tree, the “how” and “why” a solution was selected is known.

Breadth-First Search (BFS)

Summary Description:

The BFS is an uninformed search that is performed on a tree, non-cyclical graph, or a graph where a re-visit of the same state will have the same potential ensuing states. The BFS is a simple, straightforward, parallelizable method for evaluating the shortest path to a goal. In general, all nodes at a given depth in the tree are expanded before any other nodes are expanded at the next level.

Assumptions:

1. States are fully defined (i.e., it is easy to tell whether the agent has entered a given state)
2. Actions are deterministic
3. Re-visits of the same state will have the same potential ensuing state
4. World state does not change during deliberation

Problem Domains:

The BFS is widely applicable. It can be employed whenever there are sufficient computing resources.

Use Cases:

Connected networks (roads, social networks, communication networks, etc.); task allocation (requires tasks are represented as an ordered tree); any small, finite tree search where the solution must be the shortest path.

Various Extensions and Variants:

Uniform-Cost Search; Bi-Directional Search

Strengths:

BFS is provably optimal when assumptions are met.

Weaknesses:

BFS can require extensive processing time and/or computing resources. This weakness limits its applicability beyond most “toy” problems.

Opportunities for operator interaction:

Affordances:

Since the BFS is deterministic, the only opportunity that the user has to direct it is a selection of the order with which future nodes are added to the queue. Example: left-to-right instead of right-to-left.

Constraints:

None.

Agility:

None: A change during deliberation requires a full re-run of the search.

Directability:

None: Finds shortest, optimal path each time. May find different solutions depending on search ordering, but any solution found will be at the shallowest depth.

Observability:

Limited to Complete: Limited if either the states visited or pending states are made available. Complete if both of the lists are made available.

Transparency:

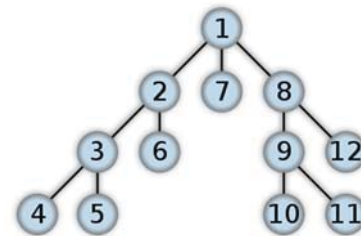
Complete: All selections are deterministic, so both the “how” and “why” a solution is found are known.

Depth-First Search (DFS)

The DFS is an uninformed search that is performed on a tree, non-cyclical graph, or a graph where a re-visit of the same state will have the same potential ensuing states. The DFS is a widely applicable search that can be used whenever there are sufficient computing resources and the goal is to find a solution with the lowest unit step cost. The DFS is a simple, straightforward, parallelizable method for evaluating the shortest path to a goal.

In order for the DFS to work as designed, the following assumptions must be true:

1. States are fully defined (i.e., it is easy to tell whether the agent has entered a given state)
2. Actions are deterministic
3. Re-visits of the same state will have the same potential ensuing states
4. World state does not change during deliberation



DFS is different from Breadth-First Search (BFS) in that it always expands the deepest unexplored node in the search tree, whereas the BFS always expands the shallowest unexplored node in the search tree. BFS searches the top of the tree first and may require much more memory than DFS because it must keep all paths from the root to the current depth within memory. DFS may require less memory, because it only has to store one path from the root node and any unexplored nodes that are the siblings/successors of the nodes on the path. The above figure shows the order in which the DFS expands the nodes. The algorithm evaluates to the deepest node in the tree along a given trajectory first. This search is neither complete nor optimal, unless it runs to completeness (full exploration of the search space), which requires the search tree to be finite. It is not optimal if it terminates prior to a full exploration of the space, because it does not guarantee the lowest path cost among all solutions. In other words, it finds a solution, but the solution might not be optimal. For example, if the tree in the above figure had solutions at nodes 6 and 8, the DFS would find the path 1-2-6 first. However, a shorter (and optimal) path is 1-8, which will be found by a BFS but not the DFS (provided the DFS terminates upon finding its first solution). Therefore, there is no way of telling which solution is optimal without searching the whole tree and finding all solutions. DFS is optimal and complete only if it does not terminate until it searches the entire search space and a solution exists.

DFS is not agile. This search cannot react to changes related to the search tree caused by environment changes. If there is a change in the search tree, the algorithm must start over since the previous tree no longer represents the current environment.

There is minimal directability in DFS. The algorithm runs as expected and will return the same solution, on the same search tree, every time. However, if the ordering of action selection is changed, a new solution may be found, which may be different from solutions found in other orderings if there are multiple solutions. Additionally, there is the possibility for more directability if there is an opportunity to continue the search after a solution is found.

The DFS algorithm accesses or holds all the relevant information to construct the complete current state of the search or the partial state of the search. If the accessed states are made available at any given time, then the searched tree is observable. If the pending list is made available, then the current state of the search is observable. Therefore, the DFS is completely observable if the accessed states and pending states are made available. Otherwise, it is minimally observable if either the accessed states or the pending states are made available.

DFS is completely transparent. Since DFS is a deterministic algorithm that will always return the same solution on the same search tree, the “how” and “why” a solution was selected is known.

Depth-First Search (DFS)

Summary Description:

The DFS is an uninformed search that is performed on a tree, non-cyclical graph, or a graph where a re-visit of the same state will have the same potential ensuing states. The DFS is a simple, straightforward, parallelizable method for evaluating a path to a goal. In general, DFS selects the deepest unexpanded node in the search tree for expansion.

Assumptions:

1. States are fully defined (i.e., it is easy to tell whether the agent has entered a given state)
2. Actions are deterministic
3. Re-visits of the same state will have the same potential ensuing states
4. World state does not change during deliberation

Problem Domains:

The DFS is widely applicable. It can be employed whenever there are sufficient computing resources.

Use Cases:

Small, finite tree searches where the solution is known to be deep within the search tree.

Various Extensions and Variants:

Backtracking Search; Depth-Limited Search; Iterative Deepening Search.

Strengths:

DFS is provably optimal when assumptions are met and the algorithm's process is not cut off when the first solution is found.

Weaknesses:

DFS can require extensive processing time and/or computing resources. This weakness limits its applicability beyond most "toy" problems.

Opportunities for operator interaction:

Affordances:

Since the DFS is deterministic, the only opportunity that the user has to direct it is a selection of the order with which future nodes are added to the queue and when to terminate the search.

Constraints:

None.

Agility:

None: A change during deliberation requires a full re-run of the search.

Directability:

Minimal: If the user is allowed to select the ordering or the user is allowed to select the termination criteria (e.g. after first or n^{th} solution found).

Observability:

Limited to Complete: Limited if either the states visited or pending states are made available. Complete if both of the lists are made available.

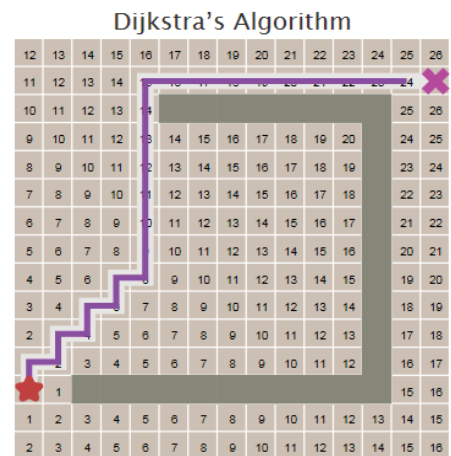
Transparency:

Complete: All selections are deterministic, so both the "how" and "why" a solution is found are known.

Dijkstra's Algorithm

Dijkstra's algorithm computes the lowest cost path from a starting node to every other node in a graph. Therefore, it will determine the lowest cost path to a goal if one exists. The figure shows that each location is labeled with the lowest cost that is required to reach that state from the starting location. If there is a goal identified, then that location will be labeled with the lowest cost as well. To get the path back to the starting location, the algorithm would travel back through the table selecting the neighboring location with lowest cost. It will continue to do this until it reaches the starting point.

Dijkstra's algorithm works well for finding the lowest-cost routes and is valuable as a preprocessing tool (before a goal is selected) in large environments. Once a goal is selected, determining the lowest-cost routes becomes trivial and can be reduced to a look-up table. Therefore, if the computing resources are available and runtime is not a factor, Dijkstra's is suited for environments that have known start points and will have many goal locations (i.e. reuse of the resulting search). However, if there is a time constraint, Dijkstra's will spend a lot of time exploring in directions that are not promising. Additionally, large environments may require too much time to perform the search and may be inefficient for the task.



Dijkstra's algorithm has limited to no directability. If the user is allowed to select a goal location, then there is directability in possibly shortening the runtime of the algorithm if the goal location is found before searching the entire search space. Additionally, the user may be given the opportunity to select the cost value for the edges (e.g. distance, fuel, time, etc.), which may change the path to the goal. Otherwise, there is no directability and the algorithm will run as programmed to find the shortest distance from the starting location to all other locations.

Since Dijkstra's algorithm accesses or holds all information needed to perform this deterministic search, the current state of the algorithm can be reconstructed at any time during the search. Therefore, Dijkstra's algorithm is completely observable when the accessed states and the pending states are made available. Dijkstra's algorithm provides limited observability into the search progress if a subset of these are made available.

Dijkstra's algorithm is completely transparent. Since Dijkstra's algorithm is a deterministic algorithm that will always return the same solution on the same search space, the "how" and "why" a solution was generated is known. Although (DFS and BFS are also completely transparent, Dijkstra's algorithm is perhaps more transparent than BFS or DFS. Dijkstra's algorithm provides a more concrete sense of "why" a route is selected, because it shows the edge costs used in its decision instead of using simple ordering of the edges for each node like DFS and BFS.

Dijkstra's Algorithm

Summary Description:

Dijkstra's algorithm computes the lowest cost path from a starting node to every other node in a graph. This will include the lowest cost path to a goal. It uses the tree search paradigm.

Assumptions:

1. States are fully defined (i.e., it is easy to tell whether the agent has entered a given state)
2. Actions are deterministic
3. Re-visits of the same state will have the same potential ensuing states
4. World state does not change during deliberation

Problem Domains:

Dijkstra's algorithm is used widely on networks and brute-force routing. It can be employed whenever there are sufficient computing resources.

Use Cases:

Path and Motion Planning

Various Extensions and Variants:

Uniform-Cost Search

Strengths:

Dijkstra's algorithm is optimal, in that the lowest cost route from a start node to any other node is calculated.

Weaknesses:

Dijkstra's algorithm has longer runtime than A*, with very little gain in memory overhead. It is best used in situations where the calculation would not have to be repeated for the entire runtime of the associated program.

Opportunities for operator interaction:

Affordances:

Adjustment in the search space, i.e., the considerations that drive edge cost, could create different routes. For example, if the edge cost is based on fuel economy, then a hybrid vehicle will have a different route than a standard gasoline vehicle. If the cost is based on time, then the routes are the same.

Constraints: None.

Agility:

None: A change during deliberation requires a full re-run of the search. A search over a subset may also be valid, but this depends greatly on the quality of the partitioning of the set.

Directability:

None to limited: The directability of Dijkstra's algorithm is limited to the search space it is searching. Cost variations and early termination are the only mechanisms by which Dijkstra's algorithm can be directed. Without being given access to these features, Dijkstra's algorithm is not directable.

Observability:

Limited to Complete: Limited if the states visited, pending states, or edge costs are made available. Complete if all are made available.

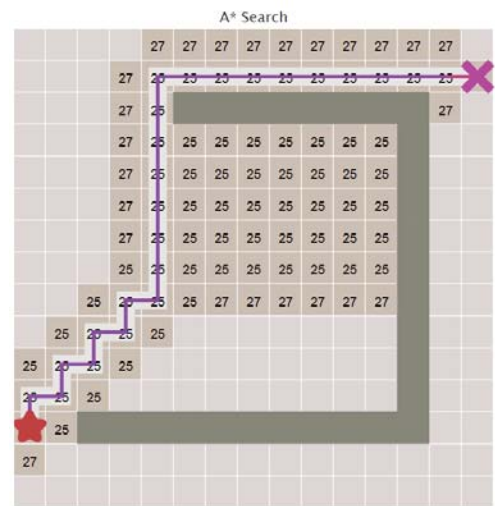
Transparency:

Complete: All reasons for a selection can be clear to a user. This is perhaps more transparent than BFS or DFS, since the edge costs provide a more concrete sense of "why" a route is selected, instead of the simple ordering of the edges for each node.

A* (A-star)

A* (pronounced A-star) is an informed search, leveraging heuristics to guide its process. It leverages the knowledge of edge cost to evaluate states. It is a reordered version of Depth-First Search (DFS), in that it always explores the lowest-valued node first (as in a priority queue instead of a stack). The heuristic for A* consists of two components: a cost to get to the current node (the only heuristic used for Dijkstra's algorithm) and an estimated cost to get from the current node to the goal. The summed value provides a node "score", where the lowest score is the next node to try. As long as the estimated cost heuristic is admissible, A* is an optimal search, and it is almost always faster and uses less memory than Depth-First or Breadth-First Searches. It is guaranteed to have the shortest runtime of any optimal algorithm. In the worst (yet unlikely) case, the runtime can be as bad as a BFS.

A* is used for many path and motion planning problems. A* finds optimal solutions given enough time to find the solution, but when the search space is vast and time constraints are limited (as in collision avoidance), the A* algorithm needs additional modifications to perform in real-time. Developers have produced updates/extensions and modifications/variants to help with pruning and heuristics for estimated cost (i.e. cost to goal). The sheer volume of extensions and variants show that there are many uses for the fundamental basis of the A* algorithm when tailored toward a specific requirement or problem set. Although A* is most widely used for motion and path planning, it can be used whenever you can measure the current cost to get to a specific state and you can create an admissible heuristic that estimates the cost to reach the goal state.



http://www.cs.cmu.edu/~cs262/lectures/a_star/introduction.html

In its most general implementation, the A* algorithm cannot react to changes within the search tree. If there is a change within the environment, the search must start over since the previous tree no longer represents the current environment. Therefore, the A* algorithm is not agile. Conversely, there are variants that allow for real-time updates and searching of small subsections of the state space when they change (e.g., D*). If implementing these variants, A* has minimal agility.

The A* algorithm has limited directability. If multiple cost heuristics are implemented and the user is allowed to select between them, then there is directability in choosing between these cost functions (e.g., time, fuel, distance, etc.). Otherwise, there is no directability and the algorithm will run as programmed to find the optimal solution based on current cost and estimated cost to goal.

Since the A* algorithm accesses or holds all information needed to perform this deterministic search, the current state of the algorithm can be reconstructed at any time during the search. Therefore, A* is completely observable when the accessed states, the pending states, and the cost function are made available. A* provides limited observability into the search progress if a subset of these are made available.

A* is a well-defined, strict algorithm. It is completely transparent as long as the cost function is understood. For example, if the user understands that the cost function is measuring/estimating distance, then the user will understand the resulting solution is the shortest distance between the start and the goal. There is no confusion regarding “why” and “how” the algorithm selected the solution.

A* (A-star)

Summary Description:

A* computes the lowest cost path from a starting node to a goal node in a graph. It employs a two-part heuristic, which includes the cost heuristic in Dijkstra's algorithm and an estimated cost to the goal. It uses the tree search paradigm.

Assumptions:

1. States are fully defined (i.e., it is easy to tell whether the agent has entered a given state)
2. Actions are deterministic
3. Re-visits of the same state will have the same potential ensuing states
4. World state does not change during deliberation
5. There are no negative cost edges (monotonic)

Problem Domains:

A* is commonly used for routing and planning. It is guaranteed to have the shortest runtime of any optimal algorithm.

Use Cases:

Path and Motion Planning (primarily); Alternatively, whenever you can measure the current cost to get to a specific state and you can create an admissible heuristic that estimates the cost to reach the goal state.

Various Extensions and Variants:

A#; Sparse A* Search (SAS); Dynamic SAS (DSAS); Dynamic A* (D*); Lifelong Planning A* (LPA); D* Lite; Delayed D*; Focused D*; Anytime Repairing A* (ARA*); Anytime Dynamic A* (AD*); Simplified Memory bounded A* (SMA*); Iterative Deepening A* (IDA*); Fringe Search; Theta*; Field D*; Beam Search; Hierarchical A*/Hierarchical Path-Finding A* (HA*/HPA*)

Strengths:

A* is optimal, in that the lowest cost route from a start node to the goal node is calculated. It is directed and thus somewhat greedy, but this is mitigated by linking the cost with the Dijkstra's algorithm component.

Weaknesses:

The runtime can be as bad as breadth-first search, but this is unlikely.

Some situations will not be applicable using A*. Example: Using A* to determine a route for an aircraft where cost is in fuel level. If the aircraft has an opportunity to refuel enroute, but has to take a longer route to get to the refueling, then this in effect creates a negative path cost. The problem would need to be reformulated to be applied.

Opportunities for operator interaction:

Affordances:

Adjustment in the search space, i.e., the considerations that drive edge cost, could create different routes. For example, if the edge cost is based on fuel economy, then a hybrid vehicle will have a different route than a standard gasoline vehicle. If based on time, the routes are the same.

Constraints:

None.

Agility:

None to minimal: In the general case, changes during deliberation requires full re-run of the search. A search over a subset may be valid, but depends greatly on the quality of set partitioning. Conversely, there are extensions/variants that do not require a re-search of the full search space, thus giving A* more agility.

Directability:

Limited: Limited to the search space and domain it is searching. Cost variations are the only mechanism by which A^* can be directed.

Limited to Complete: Limited if the states visited, pending states, and cost functions are made available. Complete if all are made available.

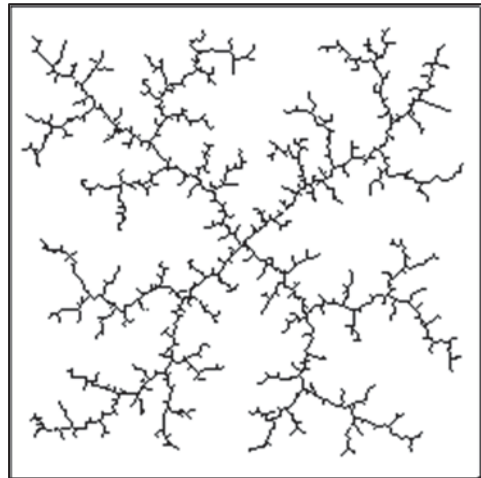
Transparency:

Complete: As long as the cost functions are understood, the “why” and “how” a solution is selected are obvious.

Observability:

Rapidly Exploring Random Tree (RRT)

An RRT is an algorithm designed by Steven LaValle to be applied to a broad class of path planning problems. It is a randomized planning technique that was specifically designed to handle the constraints of kinematics, system dynamics, and multiple degrees of freedom. It functions by randomly selecting a node, then generating a path (or portion of a path) to that node. The randomness can be biased, in that it may be directed towards the goal state, towards unexplored regions, or to a completely unbiased random state. However, the randomness factor can cause the search to spend disproportionate amounts of time not moving towards the goal. Additionally, the randomness of RRT eliminates any guarantee of optimality, runtime, or memory requirement. However, it often scales much better to larger problem spaces than Breadth-First Search (BFS), Depth-First Search (DFS), and A* since there is less that the system needs to remember regarding potential states.



Many extensions have been made to increase the efficiency and probability of connecting to the goal more quickly. Although RRTs work well for many domains, they have weaknesses that must be adjusted to handle different problem domains or classes of problems. Therefore, there are numerous extensions and variants to RRT, each intended for a particular use and specific problem domain. Since RRTs have benefits and widespread applicability to path planning, many developers and researchers have extended RRT for multiple uses within motion and path planning fields. Motion and path planning are the most suitable domains for RRT because it is difficult to sample the search space for many other domains.

In its most general implementation, the RRT algorithm cannot react to changes within the search tree. If there is a change within the environment, the search must start over since the previous tree may not represent the current environment. Therefore, the RRT algorithm is not agile. Conversely, there are variants that allow for real-time updates and searches for changes in the state space (e.g., moving objects). If implementing these variants, RRT has minimal agility.

RRT can offer limited directability. There may be the ability to bias towards goals (exploitation), unexplored regions/ Voronoi space (exploration), bounded regions, or a combination of all. However, there is no guarantee that the biasing will have a shorter runtime or a better solution, since the randomness may select other solutions or explore different regions. Routes cannot be guaranteed to be optimal nor conform to a standard, but should be satisficing.

RRT's observability is limited to the tree it is generating and the final solution. It allows the user to observe what the agent has accomplished, but gives no insight into what it is doing or will do since the selections are random. The randomness limits the transparency as well. There may be no clear reason "why" a certain portion of the route looks a certain way or "how" the route was generated (e.g. goal biasing) after RRT has generated it. However, having observability of the growth of the tree may increase the transparency of "why" or "how" the solution was selected.

Rapidly Exploring Random Tree (RRT)

Summary Description:

RRT randomly generates routes (with some bias) in order to determine a path from a start node to a goal node. It uses the tree search paradigm.

Assumptions:

1. States are fully defined (i.e., it is easy to tell whether the agent has entered a given state)
2. Actions are deterministic
3. Re-visits of the same state will have the same potential ensuing states
4. World state does not change during deliberation

Problem Domains:

RRT is best used where there is concern over the branching factor of the search. It avoids the memory overhead of holding large quantities of potential ensuing states. Since it generates routes using small extensions, it can be parallelized and thus run faster.

Use Cases:

Route, Motion, Path Planning; Multiple Degrees of Freedom Manipulation; Test case generation (limited); Automated Planning (limited).

Various Extensions and Variants:

RRT*; RRT#; RRT*-Smart; RRT-Connect; Informed-RRT*; Batch Informed Trees (BIT*)

Strengths:

Reduces some memory overhead related to potential ensuing states (resilient to high branching). Can be parallelized. Asymptotically optimal if using the RRT*.

Weaknesses:

Worst-case runtime is worse than BFS. RRTs explore slowly when the sampling domain is not well adapted to the problem.

Opportunities for operator interaction:

Affordances:

User could bias towards goal (exploitation), towards Voronoi space (exploration), or some combination.

Every iteration could be randomized (so it works one state towards the random selection), or every n iteration (so it builds some n steps towards the selection each time), or some randomization or combination of these.

Constraints:

There is a randomness factor, so it could spend a disproportionate amount of its search time focused on a part of the problem space that does not get it any closer to the goal. Except for biasing, there is no way for a user to control this.

Agility:

None to minimal: A change during deliberation requires a full re-run of the search. Minor changes can be handled, provided it doesn't eliminate a significant branch. However, there are extensions and variants that give RRT more agility by not requiring a full re-search of the space when a change (e.g. moving object) is encountered.

Directability:

Limited: There may be the ability to bias towards alternate goals, thus providing some directability on the generated route. There is no guarantee, however, since another non-preferred goal may still end up being selected. Routes cannot be guaranteed to be optimal nor conform to a standard, but should be satisficing.

Observability:

Limited to moderate: The only observability RRT offers is the growth and current state of the tree. This gives insight into possible biases and the current state of the search, but no insight into the ensuing states of the search.

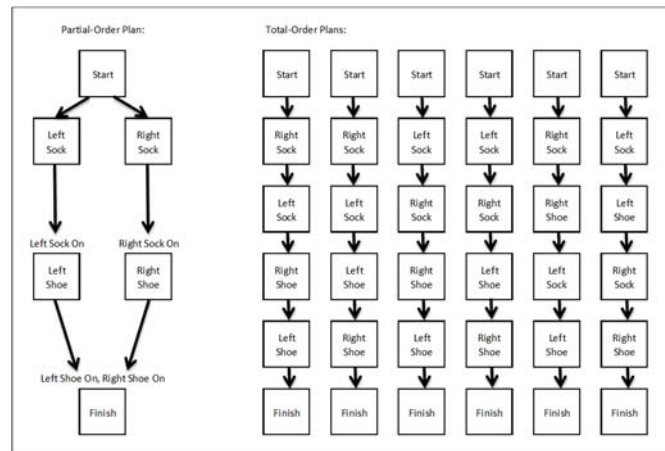
Transparency:

Limited: Because of the randomness of the iterative selection, “why” and “how” the solution is being selected are relatively known. There may be no clear reason why a certain portion of the route looks a certain way after RRT has generated it, however.

Partial-Order Planner (POP)

A (POP) is a planning algorithm that can place multiple independent actions into a plan, allowing for simultaneous or non-dependent ordering. POPs build on the constructs of Total-Order planners, by leveraging actions that require preconditions in order to activate and the result are effects on the environment. This enables a greater complexity (and possibly higher fidelity) representation of the state space that develops a sequence of actions to fulfill goal conditions.

A POP can represent the notion that actions may be independent of some things, yet dependent on others, e.g., provided the left sock is on, one can put on their left shoe, independent of the right side. The process realization is irrelevant to the POP, since it can represent the process for each foot independently. For larger problem spaces, independent actions can spawn independent solvers, aiding in parallelization and decomposition of the problem space. The figure shows that with a single representation, the POP can encompass the



six variations of the total-order planner. This makes the generated plan agnostic while enabling parallelization, supporting problem decomposition, and avoiding dependency issues. A significant drawback of state-space planners is that their formulation is sensitive. The addition, removal, or change of actions, preconditions, or effects could cause dependencies that are difficult to track down, manage, and compensate for. So, though the state-space planners leverage a potentially rich representation of the state space, they may prove difficult to scale to larger problems and have no means of handling uncertainty: all states, preconditions, and effects are deterministic. Additionally, it is difficult to estimate the heuristics, since it is not as clear how “far” a state is from the goal (i.e., estimated cost to goal state) because it does not represent states directly.

A POP’s strength is in its agility during execution. If the execution state changes such that one action is better to engage before another, then POP can handle it. However, the overall plan would need to be regenerated if any states or actions change. It is well-suited to operate in real-time, but at increased risk of inefficiency. Issues like Sussman’s anomaly and clobbering can occur because future steps may cancel the effects of previous steps.

POPs have limited directability since actions/steps have dependencies that must be upheld. Therefore, directability exists only in the ordering within nondependent actions/steps where an operator can establish ordering preferences.

POPs can provide complete observability by making available the dependencies and effects that each of the actions/steps require.

POPs have complete transparency since the resulting plan shows why the plan was selected based on the inputs and constraints of each step/action and their ensuing outputs.

Partial-Order Planner (POP)

Summary Description:

POP enables representation of simultaneous or non-dependent actions in a plan. This builds on total-order planners by leveraging a representation employing preconditions and effects for a given action. It enables a greater complexity (and possibly higher fidelity) representation.

Assumptions:

1. States are well-defined
2. Actions are deterministic
3. All effects for an action are known
4. All preconditions for an action are known
5. The state values that an action needs as a precondition do not change during the performance of the action

Problem Domains:

General planning, including robot planning. If the concept of “how” an action is executed can be separated from “which” action to use, this method is applicable.

Use Cases:

Robot task execution, constraint satisfaction, higher-level planning (e.g., making sure requirements are met prior to other activities).

Various Extensions and Variants:

ADL, PDDL, REPOP, STRIPS, UCPOP.

Strengths:

Enables a richer, yet still straightforward, representation of the state-space. The focus is on the actions, so development of a chain that fulfills the goal(s) is more natural.

Weaknesses:

The solution is tightly coupled to the state and action representation. If the representation is incomplete, or if it has any errors, then the plan will not be valid.

Even with backward solving, the search space may still be large.

It is hard to develop heuristics to guide POP.

Opportunities for operator interaction:

Affordances:

With independent sequences, there is opportunity for an operator to define ordering preferences (e.g., socks first then shoes). Users may guide action selection to compensate for incomplete state representations.

Constraints:

If an order is defined, it is immutable. Operators cannot change ordering if the planner produces a total-order subset.

Agility:

None to Limited: Not in planning, but possibly in execution. If the execution state changes such that one action is better to engage before another, then POP can handle it. However, the overall plan would need to be regenerated if any states or actions change. Well-suited to operate real-time, but increases risk of Sussman’s anomaly or clobbering (i.e., poor ordering or repeated steps).

Directability:

Limited: Directability exists only in the ordering within nondependent branches. This would enable the operator to define ordering preferences.

Observability:

Complete: All dependencies and effects of the plan’s actions can be made available.

Transparency:

Complete: How and why the approach generated a given plan is clear from the preconditions and effects of the given actions within the plan.

Markov Decision Process (MDP)

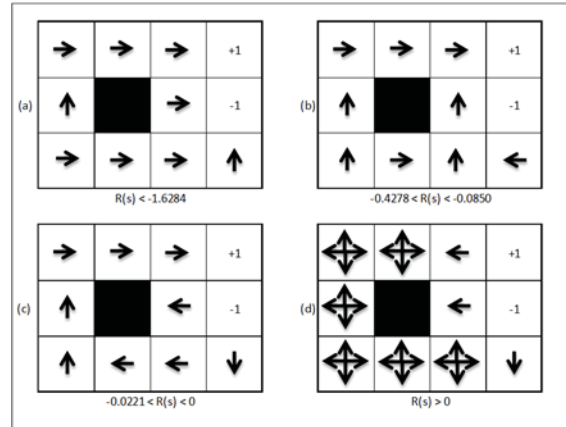
An MDP generates a policy for the state space that maximizes the probability and estimated utility of reaching a particular goal. In technical terms, a MDP is a specification of a sequential decision problem for a fully observable environment with a Markovian transition model and additive rewards. The transition model is the probabilities of each action resolving to each possible state. Markovian transition models must satisfy the Markov assumption – The current state depends only on a finite history of the previous states. The simplest Markov assumption is the *first-order Markov process*, in which the current state depends only on the previous state. If the current state sufficiently encodes the previous state, then knowledge of any previous states is unnecessary. A MDP is defined by :

Initial State: s_0

Transition Model: $T(s, a, s')$

Reward Function: $R(s)$

MDPs are widely applicable to path and planning problems, but can be used for decision making problems as long as the assumptions hold. These assumptions are: 1) The domain is fully observable where all actions resolve to state that the agent can observe and identify. 2) Actions may not be 100% effective, but the probability of success and potential resultant states is known. Probabilistic search spaces cannot have absolute solutions, since there is a chance (however small) that some actions may fail and a goal is never achieved. Thus, the MDP generates a policy denoted by π . $\pi(s)$ is the action represented by the policy π for state s . If the agent has a complete policy, it will always know which action to perform from any given state.



The goal in creating a policy is to maximize the utility of every state transition. Therefore, the value of $R(s)$ will drive the amount of “risk” the agent will accept in its policy. In the figure, (a) has a value of $R(s)$ that severely penalizes the agent for making moves. Thus, it is biased towards terminating as early as possible, even if this means a negative reward. (b) has a moderate cost for each transition; making the policy somewhat more risk averse, but still biased towards an earlier termination. (c) allows even more risk aversion, enabling the agent to take the long way around the obstacle in order to avoid the negative reward, along with avoiding the negative reward as much as possible in other states. The agent would rather try a move into the wall, leveraging the probability that it will move to safe locations on either side than to risk entering the -1 terminal state. Finally, (d) contains a reward structure that discourages termination: the policy generated ensures that the agent will never reach either of the terminal states and will continue to obtain rewards indefinitely.

MDPs require static information for transition models, reward functions, discounting and available actions. If any of these items change, the MDP values must be recalculated and the policy re-evaluated. Therefore, MDPs are not agile.

MPDs are limited to moderately directable as long as the changes are allowed within the reward functions, transition models, and discount rates. This can be used as an indirect means to affect the policy by injecting priority in speed to complete or risks of non-optimal outcomes. However, the resulting policy may not resolve to the user's intended policy.

Complete observability can be provided during agent execution, as long as the policy and current state of the agent can be provided. With these two pieces of information, the user can determine what choice the agent will take. Additionally, the user can predict the possible ensuing states if they are provided, and understand, the transition model.

MDPS are completely transparent. If given the reward function, transition model, and discount, a user can know the how and why a policy is generated. However, policy selection may appear non-intuitive based on the values given.

Markov Decision Process (MDP)

Summary Description:

MDPs focus on developing a policy for a decision space. The policy defines the best actions, given a discounted reward structure, to execute within a space. This policy is valid for stochastic actions, which are actions that may not execute as intended.

Assumptions:

1. Fully observable domain: all actions resolve to a state that the agent can observe, and place itself within.
2. Actions may not be 100% effective but their probability of success is known.

Problem Domains:

Widely applicable to path and action planning. Provided the assumptions hold, they are valid for use in decision making.

Use Cases:

Widely applicable. Representative framework supports many domains. Could support other algorithms in contingency planning.

Various Extensions and Variants:

Value Iteration, Policy Iteration, Modified Policy Iteration, Prioritized Sweeping, POMDP, Reinforcement Learning.

Strengths:

Enables handling of stochastic actions, which supports allowances for failed or erroneous actions. Stochastics are captured, so probabilities are known. Possible user directable by changing reward, transition, and discount values.

Weaknesses:

Potential infinite horizon on the plan. The state space must be represented fully, and all states in the policy must be planned. Potential workarounds exist within subspace representations.

Opportunities for operator interaction:

Affordances:

Ability to encourage a policy development that balances aggressive vs. conservative policies. Values include non-terminal rewards, transition models, discount rate.

Constraints:

The interactions between the policy and the reward function may not be intuitive. An adjacent state may have higher utility, but the policy dictates a move that does not enter that state.

Agility:

None: Although changes within the environment may not change the current policy, any changes require the MDP values and policy to be recalculated/re-evaluated.

Directability:

Limited to moderate: Limited since a user can change the values within the reward functions, which is an environmental feature. Moderate for changing transition models, and discount rates. However, this is an indirect means to affect the policy and may not result in a policy that meets the user's intent.

Observability:

Complete: States are fully observable, but actions might resolve to different ensuing states than the one intended. Therefore, the transition model, current agent state and policy must be provided for complete transparency.

Transparency:

Complete: Provided the reward function, transition model, and discount are known by the user, then transparency is present. However, policy selection may appear non-intuitive.

Localization

Localization problems are problems that deal with determining where things are located. This discussion focuses on self-localization, but the topic applies to the localization of anything. An agent leverages its known history of actions and sensor readings to perform self-localization. This aids path and motion planning since it can help to determine a starting location or course corrections to a plan if there is error in navigation and/or motion abilities. Localization is a critical aspect of planning in real-world applications, since the planning requires a valid start state.

Localization methods are typically used in conjunction with other techniques. They are used to generate probabilities of which state an agent may reside. In doing this, they could use techniques such as Hidden Markov Models (HMM), Bayesian networks, Monte Carlo localization, particle filters, Kalman filters and many other estimating techniques. Localization is a concept that has many techniques to perform the task. However, the underlying idea is that whenever movement is stochastic in a position, navigation, motion or manipulation problem, then localization techniques are used. The biggest strength of localization techniques is that they can perform with noisy sensors. However, its weakness is that it is a supporting technique and cannot perform path/motion planning directly.

Localization methods are highly agile provided the state space is known. Since the strength of this method is to estimate its current location based on sensor inputs, the method can account for changes in the environment (e.g, moving objects) as long as the environment in which it is localizing remains static. In other words, the moving objects can be incorporated as part of the noise in the sensors, but the map (e.g., walls, corridors, gaps, etc.) must be static in order to manage the uncertainty in the location estimations. Multiple estimates of the state help narrow the possible location options as new information is learned.

Localization methods provide limited directability for the user. A user may be able to provide a subset of the search space to the agent as possible, or even the exact location of the agent. However, there is no guarantee that the agent will localize to the correct location. This could happen for a number of reasons: 1) The accuracy/correctness of the sensors may identify the wrong state space. 2) The internal probabilistic models of the sensors are not correct. 3) The same (or very similar) sensor readings are in other parts of the state space, which are equally likely. 4) The state space at this location may not be feature rich enough to localize.

Localization methods can provide complete observability if the estimated locations are shown. However, there may be some stochastics in the initial particle estimates (if particle filters are in use), and this may diminish the effectiveness of the observability if there are too many possible locations. If, on the other hand, a few “very good” estimates are present, then the observability provided can be more useful/effective for the observer.

There is limited transparency in localization methods. If the user knows the algorithm, the environment and the estimates, they may be able to determine why or how the agent has determined its possible locations. Given more observability into the sensor readings and the probability models, operators may obtain a better understanding of the estimates. However, some selection mechanisms utilize stochastics, so there may be some randomness that hides the transparency even more.

Localization

Summary Description:

Localization determines the location and/or orientation of an agent within the problem space. This aids path and motion planning, as it can provide the starting point for the plan if it is not known and be used to apply course corrections during navigation.

Assumptions:

1. Agents have some sensing ability to evaluate what they see.
2. Sensors may be inaccurate, but they cannot be completely wrong all the time (since the agent depends on the sensor to determine its location).
3. All sensor error probabilities are known.
4. Some map of the state space is available.

Problem Domains:

Path planning (determination of start point, end points, or points along the path), navigation within any space.

Use Cases:

Self-localization, mapping, position verification, position tracking, global localization

Various Extensions and Variants:

Monte Carlo, Particle Filters, SLAM, Kalman Filters, Markov Localization, Bayesian Filtering, POMDP.

Strengths:

The agent can perform this work with noisy sensors.

Weaknesses:

Not a direct path/motion planner. Used to support path planning, so this will not provide path/motion planning solutions directly.

Opportunities for operator interaction:

Affordances:

Operators could direct the agent to unique locations within the space, helping resolve ambiguities.

Constraints:

Poor environmental features, initial estimates, models or sensor accuracy may prevent convergence, so no amount of operator assistance may work.

Agility:

High: Provided the state space is known, these techniques can account for moving objects by incorporating them into the noise of the sensors. Multiple estimates of the state help narrow the possible location options as new information is learned.

Directability:

Limited: A user may direct a system to localize itself within a small subset of the space or at the exact location. However, there is no guarantee that the agent will correctly localize due to poor models or the same (or very similar) sensor readings as in other states.

Observability:

Complete: The map of the environment and the agent's current estimates can be provided. There may be some stochastics in the initial particle estimates (if particle filters are in use), and this may diminish the effectiveness of the observability. If, however, a few "very good" estimates are present, then the observability can be more useful/effective for the user.

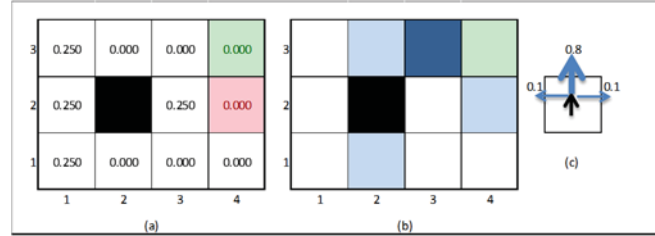
Transparency:

Limited: If the user knows the algorithm, the environment and the estimates, they may be able to determine why or how the agent has determined its possible locations. Given more observability into the sensor readings and the probability models, may increase the users understanding of the estimates. However, some selection mechanisms utilize

stochastics, so there may be some randomness that hides the transparency even more.

Partially Observably Markov Decision Process (POMDP)

Partially Observable Markov Decision Processes (POMDPs - pronounced “pom-de-pees”) contain uncertainty in both the action performance (MDPs) and the resulting state (localization problems). However, unlike localization problems, POMDPs are concerned with determining a policy based on the likelihood of which state it is in. Since the agent does not know its exact state, it cannot execute the action $\pi(s)$ recommended for state s (i.e., the policy generated in MDPs). Therefore, POMDPs must determine the most likely states and develop a policy for those possible states. Because of this, POMDPs are much more difficult than both ordinary MDPs and localization problems. Since POMDPs are by definition partially observable, we must consider what happens when we can take observations to help identify the state. A POMDP is defined by the following components:



Transition Model: $T(s, a, s')$

Reward Function: $R(s)$

Observation Model: $O(s, o)$

The transition model and the reward function are the same as in the MDP. The observation model specifies the probability of perceiving the observation o in state s . A POMDP also represents a belief state: $b(s)$. This belief state is the state estimate of the agent, represented as the probability distribution over all possible states. Therefore, $b(s)$ is the probability assigned to the actual state s by the belief state b . In other words, the belief state holds the probabilities of the “belief” that the agent is in each of the states. The belief state updates after both performing an action and taking an observation. Take the figure above, suppose the agent possesses a sensor that always detects a wall to the agent’s left, provided one is present. (a) shows the belief state of the agent in the initial state, provided it sees a wall. This belief state shows equal probability of 0.25 for each of the four locations that have a wall to the left. If the agent moves Up from this state and sees no wall to its left, (b) shows the belief state of the agent (higher probabilities have a darker shade of blue). Based on the transition model shown in (c), the current belief state shows a high probability that that the agent is in state (3, 3), but there are small probabilities that the agent is in states (2, 3), (2, 1), or (4, 2).

The fundamental insight required for understanding POMDPs is that the optimal action depends only on the agent’s current belief state. The agent’s actual state does not define the optimal action since there is no way it can know its actual state. However, the optimal action is based on the most probable state according to its belief state. By acting on the belief state, we can say that the belief state is observable and thus defines an observable MDP. Therefore, solving a POMDP on a physical state space is the same as solving an MDP on the corresponding belief state space.

POMDPs have limited agility provided the state space is known. Since its current state is a set of possible states based on observations, the method can account for changes in the environment (e.g, moving objects) as long as the environment in which it is localizing remains static. In other words, the moving objects can be incorporated as part of the noise in the sensors, but the map (e.g., walls,

corridors, gaps, etc.) must be static in order to manage the uncertainty in the location estimations. Multiple estimates of the state help narrow the possible location options as new information is learned. However, changes in the environment that affect the reward functions, transition models, discounts, and possible actions requires a re-evaluation of the policy.

Since a POMDP is a combination of MDPs and localization methods, it inherits portions of the directability, observability, and transparency of these techniques (see their respective summary sheets for more detailed explanations). Provided there is some way to indicate to the user the agent's belief state, policy, MDP functions variables, and sensor readings, then there is complete transparency. Otherwise, it may not be clear why an agent chose to do something given its state.

Partially Observable Markov Decision Process (POMDP)

Summary Description:

POMDPs combine the ability to have non-deterministic actions with non-deterministic states, thus enabling a more flexible representation. They can act as combinations of MDPs and localization.

Assumptions:

1. The state space is known and state, actions and observations are finite.
2. Error probabilities are known or can be learned.

Problem Domains:

Path planning, state estimation, policy generation

Use Cases:

Navigation, Speech Recognition, System Inspection and Maintenance, Machine Vision, Troubleshooting/ Diagnosis, Search and Rescue, Target Identification, Moving Target Search, Resource Allocation

Various Extensions and Variants:

MC-POMDP, indefinite-horizon POMDP, switching modes POMDP, decentralized POMDP

Strengths:

Enables non-deterministic actions and non-deterministic states, so can represent the search space when actions fail (or do the wrong thing) or the state estimate is inaccurate.

Weaknesses:

Assumes an infinite horizon, so there is no guarantee of termination. The full set of possible states must be known and all the probabilities for transition, observation and reward must be specified for each state, action and observation.

Opportunities for operator interaction:

Affordances:

If a user can adjust the belief state, then the POMDP may choose a different policy. Other affordances exist, as defined in MDP and localization.

Constraints:

Users cannot define some time limit, since POMDPs are not guaranteed to terminate. Other constraints exist as in MDP and localization.

Agility:

Limited: Provided the state space is known, these techniques can account for moving objects by incorporating them into the noise of the sensors. Multiple estimates of the state help narrow the possible location options as new information is learned. However, changes in the environment that affect the reward functions, transition models, discounts, and possible actions requires a re-evaluation of the policy. However, there is no guarantee that the agent will correctly localize due to poor models or the same (or very similar) sensor readings as in other states.

Directability:

Limited to moderate: Limited since a user can change the values within the reward functions, which is an environment feature. Moderate for changing the transition models, and discount rates. However, this is an indirect means to affect the policy and may not result in a policy that meets the user's intent. Additionally, limited in that users can direct POMDPs by forcing the agent to reassess its belief state at a unique point in the space or suggest its location area or exact point.

Observability:

Limited to complete: Observability is based on both the MDP structure and the localization structure. If actions resolve in an unintended manner, or if the position estimates are poor, then the observability is limited. Complete since the map of the environment and the agent's current estimates can be provided.

Transparency:

Limited to complete: Provided there is some way to indicate to the user the agent's belief state, policy, MDP function variables, and sensor readings, then there is complete transparency. Otherwise, it may not be clear why an agent chose to do something given its state.

Potential Fields

A potential field is a function defined over the state space, which defines vectors of direction and magnitude relative to the position and orientation of obstacles and goals. Potential fields provide a simple representation that might generate a path. The representation involves the goal “pulling” on the agent, and obstacles “pushing” on the agent (Figure to the right (a) and (b) respectively).

It is important to note that potential fields are defined throughout the entire state space. In practice, the potential fields are calculated for just the local space surrounding the agent (i.e., what the agent can see with its sensors). The image in the figure above shows the calculations of vectors for the entire potential field (i.e., every location) within the state space. Since potential fields only calculate the field for what it senses and does not hold state data, there is a potential to get “stuck” in local minima. Local minima occur when attractive and repulsive forces balance so the agent makes no progress or ends up in an infinite cycle. These can occur with dead end conditions or closely spaced obstacles (Figure below (a) and (b) respectively). Additionally, potential fields can pursue only one goal at a time and have no guarantee of avoiding obstacles.

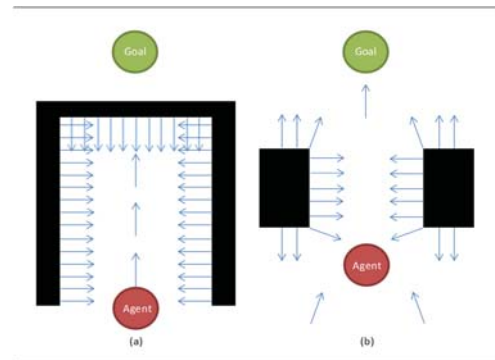
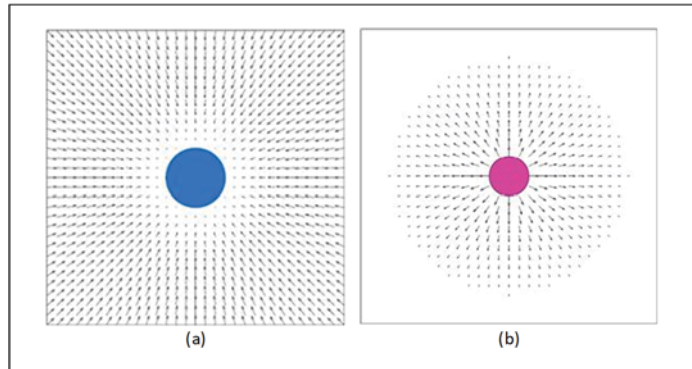
Potential fields are normally used in conjunction with other techniques to provide fast obstacle avoidance and are not used as a stand-alone path/motion planning techniques. Although local minima are serious issues that can cause an agent to get “stuck”, there are benefits to potential fields. They work well with noisy sensors, are simple to represent, and they can be used in many dynamic environments. The ease of implementing potential fields is one of the main motivations for their use. However, the vastness of the research on reducing local minima illustrates the very real, and unwanted, side effects of such a reactive technique.

Potential fields give an agent a reactive behavior to the environment. Implementing potential fields in an agent gives the agent a high level of agility in respect to changes or obstacles in the environment. No *a priori* knowledge of an obstacle’s (static or dynamic) location is required since obstacles are handled upon discovery. Therefore, the localized representation supports many other environmental changes without the need to plan extensively.

Users have limited directability with potential fields.

Users can indirectly control the agent to generate an intended behavior if they are able to add or remove obstacles, change the goal, or change the forces of the goal and obstacles.

Potential fields provide complete observability. Even with noisy sensors or poor state estimates, the responding behavior can be shown. Additionally, the force vectors can be made available. The combination of the two shows exactly what is happening regarding the agent and its behavior.



Potential fields also provide complete transparency. If the sensor data (or some representation of it) is provided with the force vectors, the user can see both why and how the agent reacted in the manner in which it reacted.

Potential Fields

Summary Description:

Potential fields generate a local solution that pushes the agent away from obstacles, and pulls the agent towards the goal. As a local solution, it benefits from a simple representation and scales well.

Assumptions:

1. The agent can sense/detect obstacles prior to collision.
2. The agent knows generally where it is in relation to the goal.

Problem Domains:

Goal-oriented path and motion planning domains.

Use Cases:

Navigation in unknown or dynamic environments.

Various Extensions and Variants:

Rotation Potential, Task Potential, Various methods to reduce/eliminate local minima (e.g. random fields, tangential fields)

Strengths:

Simple representation, robust to new information. Allows noisy sensors and/or actuators for the agents.

Weaknesses:

Since agents make decisions locally, the agent may get stuck or repeat work. No guarantees that an obstacle will be avoided. Can only pursue a single goal at any given time.

Opportunities for operator interaction:

Affordances:

Operators could add or remove virtual obstacles, or change goals. The vector sizes may be adjusted to encourage closer or more distant approaches to obstacles.

Constraints:

A goal ordering must be specifically defined by the operator, since only one goal can be pursued at a time.

Agility:

High: new obstacles are handled upon discovery, and the localized representation supports many other environmental changes without the need to plan extensively.

Directability:

Limited: Users can indirectly control the agent to generate the intended solution if they are able to add or remove obstacles, change the goal, or change the forces of the goal and obstacles.

Observability:

Moderate to complete: Even with noisy sensors or poor state estimates, the responding behavior can be shown over time. Additionally, the force vectors can be made available, which would give complete observability.

Transparency:

Complete: If the sensor data (or some representation of it) is provided with the force vectors, the user can see both why and how the agent reacted in the manner in which it reacted.

Auctioning

Auction methods are a common paradigm used to distribute tasks to agents (or agents to tasks, as the case may be). It is a distributed, decentralized approach, deferring the allocation processing to the agents within the collective. Other decentralized methods for allocating agents have been proposed and are in use today, including impatience mechanisms, model-based allocation, and hierarchical allocation. However, we specifically describe auction methods because of their saliency and their widespread use in multiagent applications. Additionally, auctions are an effective and practical approach to agent coordination because they have a small runtime, are efficient in terms of communication, calculate bids in parallel making them computationally efficient, and result in small team cost.

A general view of agent auctions is that they are not “competing” with their bids, but leveraging the bid as a communication and computation efficiency. Many variations on the auctioning paradigm are possible (some of which incorporate competitive behavior). In general, auctioning methods are used for multi-agent systems with tasks or limited resources shared across a collective, or there are parallel requirements within the same system.

As in real-world auctions, some party (in this case, an agent) acts as an auctioneer. An auction method is then engaged to allocate the task to the performing agent. The potential performing agents within the collective bid according to the auction method, and the auctioneer then awards the task to the agent with the highest (or lowest) bid. The specific auction method employed by the collective may vary: some methods emphasize flexibility, some emphasize negotiation, and some emphasize expediency of the allocation. There are three common auction types employed: parallel auctions, combinatorial auctions, and sequential auctions.

Although auctioning techniques can scale well and benefit from having lightweight representations that are straightforward to deploy, they may encounter communication bandwidth and/or error correction problems. Additionally, they are not well suited for complex tasks with prerequisites, poorly understood environments, or poorly understood collective compositions.

Auctioning provides high agility. Provided an auctioneer is made aware of it, auction methods can accept new agents and new tasks. However, there is the potential for a lower quality overall allocation when allocating a number of tasks at one time.

Users are allowed limited directability for auctions. If the user is allowed to force an allocation or set costs, then the user can target specific allocations. This forcing or target of allocations degrades from the strengths of the methods.

The auctioning process allows for complete observability. The bids during each bid cycle and final allocations can be made available, which allows the user to understand the state of the auction at any time.

Auctioning is moderately to completely transparent when the user is given the methods used to calculate the costs and to make selections. With this information and information on the bids, users can see the reasons an allocation was determined. Barring any unusual allocation or selection options related to the bidding, auctioning is completely transparent. However, auctioning is limited if the user wants to understand why an agent made a particular bid. Most auctioning methods assume that the agents define their utilities and bid so that the results of the auction serve a common objective. Therefore, transparency into the agent utility calculation is not available.

Auctioning

Summary Description:

Auctioning is a distributed, decentralized method of allocating tasks to a collective of agents. Most use a first-price bidding method, where each task is allocated based on a single round of bidding.

Assumptions:

1. Agents can determine, fairly accurately, their anticipated cost to execute a task.
2. Sufficient communication bandwidth exists for agents to trade bids.
3. Collectives are small enough that bidding is not too much of a resource load.
4. Tasks are sufficiently well-known that agents know their viability for executing a task.
5. Agents in the collective are known.

Problem Domains:

Multiagent systems with tasks or limited resources shared across a collective, or parallel requirements within the same system.

Use Cases:

Any allocation problem that has multiple resources and/or multiple tasks to be accomplished (e.g. Area surveillance, resource conservation, coordination optimization). Single agents may distribute their internal resources with this approach.

Various Extensions and Variants:

Sequential, Combinatorial, Parallel, Incremental Parallel, First-Price.

Strengths:

Lightweight representation, straightforward to deploy. Scales fairly well.

Weaknesses:

Complex tasks with prerequisites, poorly understood environments, poorly understood

collective composition. Needs good communication bandwidth, or error correction could be difficult.

Opportunities for operator interaction:

Affordances:

A user could affect the auctioneer, or could affect individual agent bids. This may drive the specific bid structure, the selection mechanism, the runtime, the resulting allocation, or the success of the task execution.

Constraints:

Too much user intervention could preclude successful execution of the ensuing tasks.

Agility:

High: Provided an auctioneer is made aware of it, auction methods are responsive to new agents and new tasks. When allocating a number of tasks at one time, however, a new task or new agent could cause a lower quality overall allocation.

Directability:

Limited: Users could target a certain allocation if they are provided the ability either to force the allocation or to set costs.

Observability:

Complete: The bids and final allocations can be made available, which allows the user to understand the state of the auction.

Transparency:

Limited to complete: Given information on the bids, users will be able to see the reasons an allocation was determined. Additionally, the users can be provided the methods used to calculate costs in order to understand the bid values. Barring any unusual allocation or selection options related to the bidding, auctioning is moderately to completely transparent depending on the user's

knowledge of the auctioning technique. However, auctioning is limited if the user wants to understand why an agent made a particular bid.

Simulated Annealing

Simulated annealing is an optimization technique used on problems where many solutions are valid, but not all solutions are desired (e.g. NP-Hard problems). Therefore, simulated annealing assumes that the domain is well suited to be represented as an optimization problem. Additionally, simulated annealing assumes the targeted minimization metric must be simple and straightforward to solve. Otherwise, additional runtime and complexity issues will be encountered since the algorithm must check the fitness of every generated solution many times.

Simulated annealing is patterned after the real-world annealing process, where heated particles in a metallic material move progressively slower as the material cools. The simulated annealing analog assumes that random guesses are made regarding the best option. As more guesses are made, the algorithm reduces the span of the random selection. Simulated annealing is a fairly simple and straightforward algorithm, designed to avoid the local minima that could be encountered when using gradient descent. It does this by balancing some of its processing time with exploration of alternatives. The simulated annealing algorithm follows these steps:

1. Choose a random solution.
2. Generate a neighbor solution.
3. Compare the two solutions, keep the one that is closer to the goal or, with some probability, the worse one.
4. Decrease temperature.
5. Repeat steps 2 to 4 until the number of iterations or the total runtime is complete.

One major drawback of simulated annealing is that despite its ability to avoid local minima, it may still spend a significant portion of its search time in a relatively poor quality portion of the search space. The single-threaded search does not consider whether the best solution found so far is on a better portion of the search space. If the annealing schedule and the random selection caused the search to “jump” outside of a certain minima, the rest of the algorithm’s work time could be spent trying to minimize in an area that cannot find a better solution. The balance of the search will be spent in the local minimum, with no improvement in the overall solution. Therefore, simulated annealing does not scale very well when compared to other optimization techniques.

Simulated annealing runs on a schedule, which defines its progress through a search. Without the ability to reset this schedule, there may be moderate agility if the minimum the algorithm is working towards is not significantly different (in terms of search space shape) from the new one. If the ability to reset the schedule is added, then it may become more agile, but not highly agile. It can respond to changes in the search space, even when a search is in progress.

Simulated annealing realizes directability in two ways, which gives simulated annealing minimal directability. The first way is the ability to change the search space such that the local minimum is similar to the operator’s intended output. The second way leverages operator input to seed the search, which may result in an output that is similar to the seeded value. This result cannot be guaranteed, however, as the annealing schedule and the random selection may direct the search to another area.

Despite the random components within the search, simulated annealing has complete observability. At any time, an operator could view the current best solution, state of the annealing

schedule, and the most recently tested solution. This is everything needed to reconstruct the state of the algorithm.

The occasional random selection of worse potential solutions diminishes the transparency of simulated annealing. Otherwise, simulated annealing has limited to moderate transparency since the annealing schedule dictates much of the decision mechanism within the algorithm. In other words, “how” simulated annealing came up with a solution is always clear due to the well-defined process, but “why” is perhaps more difficult to extract given the randomness.

Simulated Annealing

Summary Description:

Simulated Annealing is a single-threaded optimization algorithm. It leverages the real-world concepts of “temperature” and “cooling” to enable randomness and continuous improvement while performing its process.

Assumptions:

1. The domain is well-suited to be represented as an optimization problem.
2. The targeted minimization metric is simple and straightforward to solve. If this process takes too long, there may be additional runtime and complexity issues.

Problem Domains:

Optimization applications (e.g. NP-hard problems). Often these are situations where many solutions are valid, but not all solutions are desired.

Use Cases:

simulated annealing has been applied to water supply scheduling, circuit board design, structural optimization, shipping and routing (traveling salesman), image reconstruction, and even engine design.

Various Extensions and Variants:

Simulated quenching, a greedy version that is used in well-defined problems like image reconstruction. Other, similar extensions trade coverage (generalizability) for speed.

Strengths:

Provides a simple, straightforward means to solve optimization problems. The simple approach gives transparency. The general nature of simulated annealing makes it well suited to solve optimization problems across many domains.

Weaknesses:

Does not scale as well as other optimization algorithms, and is generally slower. Not

multi-threaded, so there is no straightforward way to parallelize.

Opportunities for operator interaction:

Affordances:

Operators can define the schedule, the starting and end temperature, the starting point, and perhaps even the threshold that defines when the worse choice may be accepted. This can help define the search.

Constraints:

Operators cannot guarantee that the algorithm will provide a certain solution, even with all of the affordances above. The randomness inherent in simulated annealing may direct it to a different area of the search space.

Agility:

Moderate: By default, it may or may not continue to improve even with changes in the domain depending on the current annealing schedule. However, agility can be improved if there is some means to reset the annealing schedule. This can help to ensure a better solution but does not make it highly agile since this would increase the runtime.

Directability:

Minimal: The operator can encourage a solution if they are able to define a starting point, or if they can manage the annealing schedule. However, this is not guaranteed since there is randomness in the process.

Observability:

Complete: At any time, it is easy to understand what the status of the search is - the annealing schedule dictates runtime and likelihood of random selection, which are the key aspects of the state of the algorithm.

Transparency:

Limited to moderate: There are some random processes that affect its transparency. The

opportunity to select a worse solution at random may appear somewhat opaque to operators, though this could be improved to moderate if the random selection can be visualized over time. However, since the annealing schedule dictates much of the decision mechanism within the algorithm, “how” it came up with a solution is always clear.

Ant Colony Optimization

Ant colony optimization (ACO) is an optimization technique used on problems where many solutions are valid, but not all solutions are desired (e.g. NP-Hard problems). ACO assumes the targeted minimization metric must be simple and straightforward to solve. Otherwise, additional runtime and complexity issues will be encountered since the algorithm must check the fitness of every generated solution many times.

ACO is inspired by real ant colony behavior. In real ant colonies, many ants explore randomly until a food source is found. They then return to the colony, laying a trail of pheromones on the way back. This pheromone trail guides other ants to the food source. It is a parallel algorithm, where each ant is a separate thread. The pheromones are the metaheuristic and core functional unit of the ACO approach. As with real-life pheromones, they are subject to evaporation. Thus, with paths that are followed often (and are short), the pheromones remain present and are added to (or reinforced) by other ants. For paths that are farther away or visited less, the pheromones are more likely to evaporate. The balance between exploration and exploitation occurs in ACO by defining the evaporation rate: faster evaporation means more exploration (i.e. ants do not follow preexisting paths) and slower evaporation means more exploitation (i.e. preexisting paths are followed often). ACO algorithms follow these steps:

1. Generate n ants to navigate randomly in the space, generating state-action pairs that resolve to solutions.
2. Define the goals.
3. Starting from a common location in the state space, send each ant out to generate a solution.
4. Let the ants explore randomly until they generate complete solutions.
5. When an ant generates a solution, it follows the path back to the origin, laying pheromones. These pheromones evaporate over time, and the time is something that could be defined by the developer or operator.
6. Repeat steps 3-5. For step 4, the randomness is reduced based on the strength of the pheromones. Thus, if the pheromone trail is weak, an ant may ignore it. If it is strong, it may be “forced” along the preexisting path. This probability could be tuned by a developer or operator.
7. At some point (number of iterations, amount of time, or convergence), terminate and return the optimized solution.

The ACO approach incorporates parallelization to support faster execution. However, there is no guarantee of convergence. Therefore, the termination conditions must be based on runtime requirements, which are domain specific. The parallelization inherent in ACO helps eliminate searching for disproportionate amounts of time in poorer portions of the search space. The search is less directed, so there are more opportunities to discover more optimal solutions. However, this benefit comes at the cost of more processing power since the simulated ants need to run independently. It also has a more complex representation, which reduces transparency to limited.

ACO is moderately agile. If a new target point is selected, the ants will pursue that target. Some time may be spent waiting for pheromones to evaporate, but ACO will generate new routes. Obstacles and other such aspects are handled dynamically, since the pheromone trail cannot be followed and that portion of the route will evaporate.

To improve operator acceptance and to make ACO more directable, there may be a possibility of seeding the solutions: either biasing some of the ants' first paths, or predefining a pheromone level. This can direct the solution to produce an intended path, though there is no guarantee that the path will remain as directed given the stochastic nature of the algorithm. Other directability exists with the pheromone evaporation rate, which prefers exploitation vice exploration.

ACO can be observable if the ants and/or the pheromone trails and their strengths are visualized. This makes it is easy to understand the state of the algorithm. Since the "how" ACO came up with a solution is always clear due to the well-defined process, the "why" is more difficult to extract given the randomness. Therefore, this visualization will increase its transparency from limited to moderate.

Ant Colony Optimization

Summary Description:

Ant colony optimization (ACO) mimics the behavior of real-world ants. After discovering a valid solution, these ants lay pheromones on the way back. Over time, pheromones evaporate. Shorter paths are reinforced more often, because ants travel them more frequently and reinforce them at each pass. The strongest pheromone trail is generally the shortest known path.

Assumptions:

1. The domain is well-suited to be represented as an optimization problem.
2. The targeted minimization metric is simple and straightforward to solve. If this process takes too long, there may be additional runtime and complexity issues.

Problem Domains:

Optimization applications (e.g. NP-hard problems). Often these are situations where many solutions are valid, but not all solutions are desired.

Use Cases:

Use cases are numerous. Applications have included scheduling, vehicle routing, set coverage, circuit design, image processing, data mining, classification, network routing, and protein folding calculations.

Various Extensions and Variants:

Many variants exist that utilize different metaheuristics for laying and weighting the pheromone trail. Other variants focus on parallelizing the search space.

Strengths:

Provides a natural, low-overhead means to solve optimization problems. The ants show clear behaviors related to exploration and exploitation, enhancing observability. Parallelization helps to accelerate the

solution finding. It is well suited to solve optimization problems across many domains.

Weaknesses:

Very sensitive to the domain. The evaporation rate controls the exploration/exploitation tradeoff, but its sensitivity is not clear for all domains. There is no clear termination point, since a random ant may find a significantly better route at any time. The termination point needs to be dictated by the operator.

Opportunities for operator interaction:

Affordances:

Operators can define the evaporation rate (exploration/exploitation tradeoff), the number of ants (runtime/computational resources tradeoff), the probability at which ants deviate from established paths, and the runtime (to stop the process).

Constraints:

There is no natural automatic termination, so it must be defined in some way. This may limit the quality of the solution.

Agility:

Moderate: New domain information may change the generated routes, provided there is a probability of an ant exploring the new space. When a path has been highly reinforced, ants may not tend to deviate, so there may need to be some artificial method to degrade the solution.

Directability:

Minimal: The operator can encourage a solution if they are able to define a starting point, or if they can manage the pheromone evaporation rate. This is not guaranteed, however, since there is randomness in the process.

Observability:

Limited to complete: the current state of the pheromones and the ant status can be generated. Often, these are shown on a map with the current best solution, so observability is more explicit and complete.

Transparency:

Limited to moderate: The occasional random path selection by the ants diminishes the transparency. If the pheromone trails are shown, then the reasons why most choices have been made are clear.

Genetic Algorithms

Genetic Algorithms (GAs) are optimization technique used on problems where many solutions are valid, but not all solutions are desired (e.g. NP-Hard problems). GAs assume the targeted minimization metric must be simple and straightforward to solve. Otherwise, additional runtime and complexity issues will be encountered since the algorithm must check the fitness of every generated solution many times.

GAs are inspired by real-world analogs. In a GA, the metaheuristics generate genes through various processes (crossover, mutation, elitism, etc.), which are then passed through a heuristic to measure the quality of the gene. This measurement is called a fitness function. The genes can be ranked and then combined, creating offspring. This natural selection process (exploitation) is balanced using mutations, where a random subset of the gene is swapped for a randomly generated value or set of values (exploration). GAs are inherently parallelized.

There are a number of factors to consider when employing a GA:

1. Gene construction: This is the gene's representation. Many representations could be viable, but any given gene would need to represent a possible, complete solution. Typically, the gene is represented as a string over a finite alphabet – most commonly a string of 0's and 1's.
2. Fitness function: This function translates a gene's values into a solution, and then measures the quality of that solution. This is used to rank genes against each other.
3. Mutation rate: This is a measure of the probability of a gene receiving a random change to one or more of its values. This dictates how much the GA may explore at any given time. This could be per gene or per value.
4. Selection: This defines the selection mechanism to use when selecting genes from the population to combine. Selection may also dictate "cloning" members to the next generation (i.e., elitism).
5. Crossover: The process of combining two parent genes (A and B) to produce an offspring.

The construction of GAs is such that there are many opportunities for operator interaction. The number of generations, along with the population size, dictates the runtime of the algorithm. The operator can have control of either or both of these values. If given an opportunity to do so, the operator may be able to define the fitness function, which determines the GA's target optimization. To shift the balance between exploration and exploitation, the operator may define the mutation rate. This rate could be on a schedule, which could encourage more mutation (more exploration) early in the runtime and less mutation (more exploitation) later.

GAs are moderately to highly agile: the fitness function defining the quality of a given gene can be changed to respond to problem space changes, and the GA can then optimize to it. However, if the GA has been running for a significant time, much of the population may be optimized for the preexisting fitness function. In this case, more time would be needed to compensate for the new fitness function. The fitness function can be changed at any time, with no other changes except for runtime.

GAs can provide a user minimal directability with the use of a seeding function. The initial population or a subset of its members could be seeded to guide the search and provide an output similar to what an operator intends. However, given the stochastic nature of the mutation and crossover behavior, this direction cannot be guaranteed.

GAs have limited observability since the state of the GA is the population at any given iteration. As long as the population can be extracted, there is no problem with understanding the state. However, this merely helps to define a percentage of the search completion, not observability into the algorithm's state.

A GA has limited transparency. With the exception of knowing what the GA is optimizing against (the fitness function), it is often not clear how the GA generated a certain gene. If a random mutation created the one value that extended a gene to have greater fitness value, then other genes should begin to follow the same pattern. However, there is no clear way to know whether the cause of that fitness was from some crossover or some mutation. In ensuing generations, those updates could be eliminated by a random process, and the fact that the GA was improving is lost.

Genetic Algorithms (GA)

GAs mimic the reproductive and mutation behavior of real-world genetics. They leverage the concept of crossover (reproduction) and mutation (random change) to balance exploration and exploitation during their search. Multiple genes are created, and these genes represent solutions.

Assumptions:

5. The domain is well-suited to be represented as an optimization problem.
6. The targeted minimization metric is simple and straightforward to solve. If this process takes too long, there may be additional runtime and complexity issues.

Problem Domains:

Optimization applications (e.g. NP-hard problems). Often these are situations where many solutions are valid, but not all solutions are desired.

Use Cases:

Use cases are extensive. Applications have included all those listed for simulated annealing and ant colony optimization, along with learning walking behaviors, antenna design, and training ANNs. The use case list is too extensive to list here.

Various Extensions and Variants:

Many variants exist that utilize different metaheuristics for selection, crossover, elitism, and mutation.

Strengths:

Can solve many optimization problems much more efficiently than other standard approaches. With the ability to tune the population size and number of generations, operators can control runtime and computational load directly.

Weaknesses:

There is no straightforward way to determine if the GA has found the optimal solution, so the termination conditions are based on population size and number of generations. This may not provide a satisficing solution.

Opportunities for operator interaction:

Affordances:

Operators can define population size, mutation rate, crossover rate, and number of generations. This controls runtime, exploration versus exploitation tradeoff, and computational resource needs.

Constraints:

There is no natural automatic termination, so it must be defined in some way (usually through number of generations). This may limit the solution's quality.

Agility:

Moderate to high: New domain information may change the fitness results of a given gene, changing its ranking. If there is a schedule to the mutation rate (where it diminishes as the runtime extends), there would need to be a way to reset it. Otherwise, the agility is extensive and is provided implicitly.

Directability:

Minimal: The operator can encourage a solution if allowed to define a starting set of genes, or if they can manage the mutation rate. However, it is not guaranteed to produce a better solution due to the randomness of selection, crossover, and mutation.

Observability:

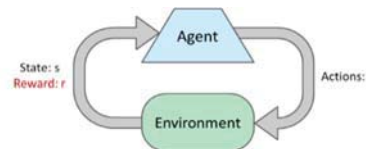
Limited: The current state of the genes can be provided, but it does not provide much information related to the search process. The number of generations evaluated thus far can be used, but this merely helps to define a percentage of the search completion, not observability into the algorithm's state.

Transparency:

Limited: A single mutation may significantly improve or degrade a gene's fitness. Since a given gene is only one of many possible solutions, it is difficult to extract how the GA created its final recommended solution. The "why" aspect of the solution is straightforward from an evaluation standpoint, provided an operator knows the fitness function. It is not straightforward from a selection standpoint, given the random selection and perhaps large population size.

Reinforcement Learning

The basic idea for reinforcement learning (RL) is that the agent must act within the environment and observe the feedback it receives in order to determine how to act in the future. In other words, the agent receives feedback in the form of rewards (could be positive or negative), determines its utility based on a reward function, then “learns” how to act to maximize the expected rewards. This context is much like the Markov Decision Process (MDP), but MDPs already know both the transition model and reward function. Thus, the MDP agent determines the optimal action policy without interacting with the environment because it has all the information it needs. When the agent is not given the transition model and/or the reward function, it must act within the environment to determine (i.e. learn) these values. The agent still works to determine the appropriate policy, but it must gain experience to do so by acting in the environment, working online. Another way to look at the difference between offline solutions and online learning is that in offline solutions, “bad” states do not need to be experienced to know they are bad, since the information is already available in the form of the reward function. In online learning, the agent must enter “bad” states and obtain the associated reward to determine that the state is bad.



There are many techniques to use for RL. Some try to learn a specific model (model-based). Some try to mimic the Bellman Update Equation (shown to the right), which is used in solving MDPs. Some try to generalize the state into features representation, which is used for complex and partially known environments. These feature representations can be used to write a function that contains weights on each feature. The function can represent any state using the weights and the features, as shown in the approximate Q-value function shown to the left. Many of the RL techniques follow a general philosophy in making updates to the learned values after each observation. They incorporate the use of a learning rate (typically denoted as alpha - α) by adjusting the estimated value of the state in the direction of the error by size alpha. A high alpha value learns faster, but a small alpha value balances the learning across many experiences. The risk of overfitting increases with higher alpha values and underfitting increases with lower alpha values. Additionally, all learning techniques must balance exploration with exploitation.

$$V_{k+1}^{\pi}(s) \leftarrow \sum_{s'} T(s, \pi(s), s') [R(s, \pi(s), s') + \gamma V_k^{\pi}(s')]$$

Since RL requires that agent to learn within the environment through experiences and feedback, the agent will learn the new environment if it changes. This gives RL techniques a high level of agility.

The elements of directability depend on the RL technique used. RL allows for directing the learning rate, the amount of exploration vs. exploitation, and the training that the agent may experience. This allows the user complete directability in the RL process, but it does not guarantee that it will learn the intended behavior because of the stochastic nature of the environment and agent actions.

The observability of the RL process is limited to complete. Since the agent must interact with the environment to receive the rewards and update their estimated values, RL can allow complete observability into these values. However, these may not give the observer much insight into the

learning state since the number of training episodes may not be available. Many methods do not, or cannot, keep track of the number of times a state has been visited or the quality of the experience. Therefore, the user will not know how much of the state has been sufficiently learned. This, along with the operator's knowledge of the RL technique used, limits the transparency that RL can provide. The user will understand how the agent is learning but not why it learned some behavior and not another.

Reinforcement Learning

Summary Description:

In reinforcement learning (RL), the agent acts in the environment and receives feedback in the form of rewards (could be positive or negative) that it uses to determine the utility of that action given the state. It uses this information to “learn” an optimal (or nearly optimal) policy that will maximize the expected rewards for that environment.

Assumptions:

1. The rewards for the same state-action pair are consistent.
2. The rewards give an indication if an action is “good” or “bad”.
3. The learning agent has access to the environment to experiment and learn.

Problem Domains:

Any domain that has a finite set of actions/decisions where an agent must learn how an environment works

Use Cases:

Use cases are numerous. In general, RL is used when an agent does not know how the environment works and must learn to act successfully in the environment.

Various Extensions and Variants:

Passive, active, model-based, model-free, Q-learning, Approximate Q-learning, state-based representation

Strengths:

It can be used on many complex problems from very structured and known environments (model-based learning) to complex and partially known environments (state-based representation).

Weaknesses:

Must have some way of acting in the environment and experiencing the rewards.

Opportunities for operator interaction:

Affordances:

Operators can set the learning (alpha) rate. A high alpha value learns faster, but a small alpha value balances the learning across many experiences. Operators can adjust the amount of random actions to take (epsilon) to control explorations vs. exploitation. The operator may be able to choose which actions the agent takes to direct specific actions to learn.

Constraints:

The risk of overfitting increases with higher alpha values and underfitting increases with lower alpha values. If the epsilon value is never decreased, the agent will not execute the learned behavior. Changes in the environment may require additional training.

Agility:

High: RLs learn within the agent’s environment. If the environment changes, the agent’s expected values will change as it experiences more of the changed environment.

Directability:

Complete: the elements of directability depend on the technique used. RL allows for directing the learning rate, the amount of exploration/exploitation, and what actions the agent takes.

Observability:

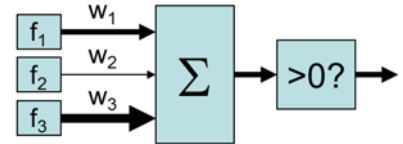
Limited to complete: since the agent must interact with the environment to receive the rewards and update their estimated values, RL can allow observability into these values. However, these may not give the observer much insight into the learning state since the number of training episodes may not be available.

Transparency:

Limited: depending on the technique used, transparency is limited to the operator's knowledge of the technique and how much of the training has occurred. This limitation may also be based on the quality of the training experiences.

Artificial Neural Network (ANN)

Artificial Neural Networks (ANNs) use a bio-inspired neural network structure. The structure is composed of nodes (perceptrons) with activation functions and edges with weights. The edges feed values to the nodes, which activate according to their activation functions, possibly feeding associated values along their output edges. With an artificial neural network (ANN), the state is represented by the inputs, the neural network's structure represents the model, and the decision is the output. The ANN learns by changing the internal weights. In other words, instead of having a rigorous representation (like the MDP), the ANN can simultaneously learn the input-output relationships AND generate a model for the domain. The domain model is maintained implicitly within the network structure. The simplest ANN is a perceptron, which is a computational model of a single neuron in a human brain. In its simplest form, a perceptron uses a feed-forward model, where it takes one or more inputs, processes those inputs, and generates an output without sending any information back to the inputs or to itself.



The inputs to the perceptron are feature values (f_x) and each feature value has a weight (w_x). After the weights are applied to the input/feature values, they are fed into an activation function. Activation functions are widely varied and range from very simple to very complex. They can be continuous (e.g. regression) or discontinuous (e.g. classification). These functions include binary, radial basis, sigmoid, and linear functions. In most traditional cases, the activation functions should be differentiable. This makes training more straightforward, as the error can be compensated for by following a gradient. The activation function shown in the figure is the sign of the sum (Σ) of all weighted inputs. If the sign of the sum is positive, the output is +1. If the sign of the sum is negative, the output is -1. Extension of a perceptron to an ANN simply requires connecting multiple perceptrons together.

The greatest benefit of ANNs is that they can serve as the domain model according to the agents. The input-output mapping is determined from the structure and weights, so the unknown function that describes the domain is approximated without direct knowledge of the function. The domain model reflected in the ANN network structure can be well-captured, provided the training data is sufficiently diverse and representative. The flexibility and diversity of ANN representations are both a great strength and a great weakness. Since some network structure can exist for any continuous function, ANNs can solve an extensive set of problems. However, the structures and the relationship between the nodes and weights make them difficult to interpret. Cross-dependencies are well handled by the structure, but are not easily extracted. Human understanding of larger ANNs is seriously limited. Because of this, ANNs are generally considered quite opaque, which leads to difficulty in interpretation, verification and validation, and scoping. This opacity may lead to difficulties in directability and observability. However, the gains in agility may be worth sacrificing these features, since the ability to restructure the ANN and for it to learn from new information could provide a critical level of agility that cannot be achieved by any other known method.

If the network is underfit or overfit, its agility may be severely reduced or non-existent. Otherwise, ANNs can be highly agile. New training data may change some weights, which change the resulting output. Depending on the domain, the ANN may already handle the new data without training.

ANNs can provide complete observability, however, interpretation may be limited. If the network structure and the associated weights can be shown, then the state of the ANN can be extracted at any point in a straightforward manner. However, the meaning (or transparency) of the network structure and how it shapes inputs to outputs is still limited. The only way to enhance transparency is to reduce the size of the neural network such that it is easily interpretable by an operator. With networks this small, however, there is often a severe limitation in the ANN's ability to handle certain events due to underfitting.

Artificial Neural Network (ANN)

ANNs use a bio-inspired neural network structure. The structure is composed of nodes with activation functions and edges with weights. The edges feed values to the nodes, which activate according to their activation functions, possibly feeding associated values along their output edges.

Assumptions:

1. The target output is either a classification value (e.g. binary true/false) or continuous.

Problem Domains:

Problem domains are vast. They have included stock market prediction, character recognition, optimization, play selection in football, self-driving cars, dynamic control, perception, and decision making to name a few. The domain list is too extensive to include here.

Use Cases:

Use cases are numerous. There is a way to apply an ANN to most known computational and real-world problems. None are listed, as the list is excessively long.

Various Extensions and Variants:

Feedforward, recurrent, LSTM, NEAT, EANT, Kohonen, RBF, LVQ, LSTM etc. Too numerous to list here.

Strengths:

Applicable to a vast number of problems. Many network structures provide many variations. Therefore, many networks can solve the same problem and individual networks can solve multiple problems.

Weaknesses:

Will only change behavior if given new training data, which may take some time or could eliminate current “good” behavior. Is

only as good as its training set. May become underfit or overfit.

Opportunities for operator interaction:

Affordances:

Operators can be given ability to deliver or adjust training data. They can act in a supervisory role to guide learning. During operation, the operator may be able to intervene, which could define a re-training of the ANN. This intervention depends on the domain, and drives training in all cases. Otherwise, affordances only exist to override the ANN behavior.

Constraints:

The quality of fit is dictated by the ANN structure and training data. A poor structure (relative to the domain) or poorly representative training data will greatly limit this quality. No operator intervention may be able to compensate for these limitations.

Agility:

None to high: if the network is underfit or overfit, its agility may be severely reduced or non-existent. Otherwise, ANNs can be highly agile. New training data may change some weights, which change the resulting output. Depending on the domain, the ANN may already handle the new data without training. Thus, a well-formed, well-trained ANN may be more agile than other known techniques.

Directability:

None to Complete: Directability depends on the training data and the network structure. For some networks, the user can direct it to learn the intended output and some it will not be possible at all.

Observability:

Limited to complete: Every aspect of the network can be made observable, however, interpretation may be limited. If a graph of the network structure and the associated weights can be shown, then the state of the ANN can be extracted at any point in a straightforward manner. However, the meaning of the network structure and how it shapes inputs to outputs is still limited, even with this observability feature.

Transparency:

Limited: Transparency in ANNs is severely limited. The only way to enhance transparency is to reduce the size of the neural network such that it is easily interpretable by an operator. With networks this small, however, there is often a severe limitation in the ANN's ability to handle certain events due to underfitting.

LIST OF ABBREVIATIONS AND ACRONYMS

ACO	Ant Colony Optimization
AD	Dynamic A*
ADOT	Agility, Directability, Observability, Transparency
AFRL	Air Force Research Laboratory
AI	Artificial Intelligence
ANN	Artificial Neural Network
ARA*	Anytime Repairing A*
BFS	Breadth-First Search
COATDS	Course of Action Tools for Decision Superiority with Autonomous Systems
D*	Dynamic A*
DFS	Depth-First Search
DoD	Department of Defense
DSAS	Dynamic Sparse A* Search
EA	Evolutionary Algorithms
HA*/HPA*	Hierarchical A*/Hierarchical Path-Finding A*
IDA	Iterative Deepening A*
IEEE	Institute of Electrical and Electronics Engineers
GA	genetic algorithms
HMI	Human Machine Interface
LPA	Lifelong Planning A*
MDP	Markov Decision Process
PDDL	Planning Domain Definition Language
POMDP	Partially Observable Markov Decision Processes
POP	Partial-Order Planner
RL	reinforcement learning
RRT	Rapidly-Exploring Random Trees
RRT*FN	RRT* Fixed Node
SAA	Sense and Avoid
SAS	Sparse A* Search
SLAM	Simultaneous Localization and Mapping

SMA*	Simplified Memory Bonded A*
STRIPS	STanford Research Institute Problem Solver
US	United States