Artificial Intelligence and Automation

Sven Koenig, Shao-Hung Chan, Jiaoyang Li, Yi Zheng

Abstract In this chapter, we discuss Artificial Intelligence (AI) and its impact on automation. We explain what AI is, namely the study of intelligent agents, explain a variety of AI techniques related to acquiring knowledge from observations of the world (machine learning), storing it in a structured way (knowledge representation), combining it (reasoning), and using it to determine how to behave to maximize task performance (planning), in both deterministic and probabilistic settings and for both single-agent and multi-agent systems. We discuss how to apply some of these techniques, using automated warehousing as case study, and how to combine them. We also discuss the achievements, current trends, and future of AI as well as its ethical aspects.

Key words: Artificial Intelligence, Automated Warehousing, Ethics, Intelligent Agents, Knowledge Representation, Machine Learning, Multi-Agent Systems, Optimization, Planning, Reasoning

Sven Koenig
USC, e-mail: `skoenig@usc.edu`

Shao-Hung Chan
USC, e-mail: `shaohung@usc.edu`

Jiaoyang Li
USC, e-mail: `jiaoyanl@usc.edu`

Yi Zheng
USC, e-mail: `yzheng63@usc.edu`

Contents

List of Symbols and Abbreviations

AI	Artificial Intelligence: the study of intelligent agents
CNN	Convolutional Neural Network: a specialized NN
CSP	Constraint Satisfaction Problem: the problem of assigning values to variables so that given constraints are satisfied
FOL	First-Order Logic: a language for knowledge representation
GA	Genetic Algorithm: a local search technique
GAN	Generative Adversarial Network: a machine learning technique
iid	independent and identically distributed: a sampling assumption
ILP	Integer Linear Program: a specification language for function optimization
LASSO	Least Absolute Shrinkage and Selection Operator: a regression technique
LP	Linear Program: a specification language for function optimization
LSTM	Long Short-Term Memory: a way of storing state in RNNs
MDP	Markov Decision Process: a specification language for probabilistic planning
MILP	Mixed Integer Linear Program: a specification language for function optimization
NN	Neural Network: a machine learning technique
PCA	Principal Component Analysis: a machine learning technique
PDDL	Planning Domain Definition Language: a specification language for deterministic planning
POMDP	Partially Observable MDP: a specification language for probabilistic planning
PPDDL	Probabilistic PDDL: a specification language for probabilistic planning
RL	Reinforcement Learning: a stochastic dynamic programming technique for solving MDPs
RNN	Recurrent Neural Network: a specialized NN
SAT Problem	Satisfiability Problem: the problem of finding an interpretation that makes a given propositional sentence true
STRIPS	Stanford Research Institute Problem Solver: an early planner for deterministic planning but now used for the specification language of that planner
SVM	Support Vector Machine: a machine learning technique
TSP	Traveling Salesperson Problem: the problem of finding a shortest route that visits each city of a given set of cities exactly once and returns to the start city

1 Artificial Intelligence (AI): The Study of Intelligent Agents

According to the Merriam-Webster dictionary, *intelligence* is 1) "the ability to learn or understand to deal with new or trying situations" or 2) "the ability to apply knowledge to manipulate one's environment or to think abstractly as measured by objective criteria (such as tests)" [1]. The field of *Artificial Intelligence* (AI) studies how to endow machines with these abilities.

Compatible with this definition, creating AI systems is often equated with building (intelligent) agents, where an *agent* is an input-output system that interacts with the environment, similar to a feedback controller. It receives observations about the state of the environment from its sensors and can execute actions to change it. How to obtain high-level observations from low-level sensor signals (for example, with vision, gesture recognition, speech recognition, and natural language processing) and how to translate high-level actions into low-level effector signals is considered part of AI but not of core AI and thus not discussed here. The program of an agent is essentially a function that specifies a mapping from possible sequences of past observations and actions to the next action to execute. *Cognitive agents* use functions that not only resemble those of humans but also calculate them like humans. In other words, cognitive agents "think" like humans. In general, however, agents can use any function, and the function is evaluated according to a given objective function that evaluates the resulting agent behavior. For *believable agents*, the objective function measures how close the agent behavior is to that of humans. In other words, believable agents behave like humans, which might include modeling human emotions. Examples are intelligent voice assistants and lifelike non-player characters in video games. For *rational agents*, the objective function measures how well the agent does with respect to given tasks. In other words, rational agents maximize task performance. Examples are chess programs and autonomous vacuum cleaners.

AI was originally motivated mostly by creating cognitive agents since it seems reasonable to build agents by imitating how existing examples of intelligent systems in nature, namely humans, think. AI quickly turned to non-cognitive agents as well since the computing hardware of agents is different from the brains of humans. After all, it is difficult to engineer planes that fly by flapping their wings like existing examples of flying systems in nature, namely birds, due to their different hardware. For a long time, AI then focused on rational agents, mostly autonomous rational agents without humans in the loop but also autonomous rational agents that are able to obtain input from humans, which often increases the task performance, and non-autonomous rational agents (such as decision-support systems). More recently, AI has also started to focus on believable agents as technological advances in both AI and hardware put such systems within reach.

Both rational and believable agents are important for automation. Rational agents can be used to automate a variety of tasks. Believable agents offer human-like interactions with gestures and speech and understand and imitate emotions. They can, for example, be used as teachers [44, 66] and companions [138, 24] as well as for elderly care [113, 135, 26] and entertainment purposes [137]. In this article, we focus on rational agents. A thermostat is a rational agent that tries to keep the temperature close to the desired one by turning the heating and air conditioning on and off, but AI focuses on more complex rational agents, often those that use knowledge to perform tasks that are difficult for humans.

2 AI Techniques

Knowledge is important since rational agents make use of it. The initial emphasis of AI on cognitive agents explains why the study of rational agents is often structured according to the cognitive functions of humans. One typically distinguishes acquiring knowledge from observations of the world (*machine learning*), storing it in a structured way (*knowledge representation*), combining it (*reasoning*), and using it to determine how to behave to maximize task performance (*planning*). The interaction between agents is also important. We will discuss all these AI techniques separately in the context of stand-alone examples but also how to combine them. For example, one can use knowledge representation to represent chemistry knowledge and reasoning to utilize this knowledge to determine what happens when one mixes two substances together. If one does not know which substances oxidize, then one can perform experiments with some substances and use machine learning to predict for untested and even not-yet-synthesized substances whether they will oxidize. Finally, one can use planning to come up with a sequence of steps to synthesize new substances with given properties cheaply and, in case multiple persons are needed for this task, specify who does what, where, and when. Rational agents can be built based on a variety of techniques, often related to optimization, since they maximize task performance. Since trivial techniques are typically much too slow, AI develops techniques that exploit problem structure to result in the same or similar agent behavior but satisfy existing time constraints. Some of these techniques originated in AI but others originated much earlier in other disciplines that have also studied how to maximize performance, including operations research, economics, and engineering. AI has adopted many such techniques. The techniques used in AI are therefore very heterogeneous.

2.1 Optimization

Optimization is the task of assigning values to variables, possibly under some constraints, so as to minimize the value of a function of these variables. One often cannot systematically enumerate all possible assignments of values to all variables (*solutions*) and return one with the smallest function value because their number is too large or even infinite. In this case, one often starts with a random solution and then moves from solution to solution to discover one with a small function value (*local *search*). Optimization techniques allow planning to maximize task performance and machine learning to find good models of the world.

2.1.1 Continuous Function Minimization

The *continuous function minimization problem* is to find a global minimum of a function of several variables with continuous domains. (A global maximum can be determined by finding a global minimum of the negative function.) Since this can be difficult to do analytically, one often uses local search in form of gradient descent to find a small local minimum instead. *Gradient descent* first randomly assigns a value to each variable and then repeatedly tries to decrease the resulting function value by adjusting the values of the variables so that it takes a small step in the direction of the steepest downward slope (against the gradient) of the function at the point given by the current values of all variables. In other words, it first chooses a random solution and then repeatedly moves from the current solution to the best neighboring solution. Once this is no longer possible, it has reached a local minimum. it repeats the procedure for many iterations to find local minima with even smaller function values (*random restart*), a process which can easily be parallelized, until a time bound is reached or the smallest function value found is sufficiently small. Gradient descent can use a *momentum term* to increase the step size when the function value decreases substantially (to speed up the process) and decrease it otherwise.

Function minimization under constraints has been studied in AI in the context of constraint programming but has also been studied in operations research. A continuous function minimization problem under constraints can be formulated as follows:

$$\begin{aligned} \underset{x_1,\ldots,x_n \in \mathbb{R}}{\text{minimize}} \quad & f_0(x_1,\ldots,x_n) \\ \text{subject to} \quad & f_i(x_1,\ldots,x_n) \le 0, \text{ for all } i \in \{1,\ldots,m\}. \end{aligned} \tag{1}$$

Here, $x_1,\ldots,x_n \in \mathbb{R}$ are real-valued variables, f_0 is the function to be minimized (*objective function*), and the remaining functions f_i are involved in constraints (*constrained functions*). (An equality constraint $f_i(x_1,\ldots,x_n) = 0$ can be expressed as $f_i(x_1,\ldots,x_n) \le 0$ and $-f_i(x_1,\ldots,x_n) \le 0$. Similarly, a range constraint $x_j \in [a,b]$ can be expressed as $-x_j + a \le 0$ and $x_j - b \le 0$.) Lagrange multipliers can be used for function minimization if all constraints are equality constraints. Other special cases include *Linear Programming* (LP), where the objective function and constrained functions are linear in the variables, and *quadratic programming*, where the objective function is quadratic and the constrained functions are linear [14]. "Programming" is a synonym for optimization in this context.

2.1.2 Discrete Function Minimization

The *discrete function minimization problem* is to find the global minimum of a function of several variables with discrete domains. The local search analog to gradient descent for discrete functions is hill climbing. One has to decide what the neighbors of a solution should be. Their number should be small compared to the number of solutions. Like gradient descent, *hill climbing* first chooses a random solution and then repeatedly moves from the current solution to the best neighboring solution. Unlike gradient descent, hill climbing can determine the best neighboring solution simply by enumerating all neighboring solutions and calculating their function values.

For example, the *Constraint Satisfaction Problem* (CSP) consists of variables, their domains, and constraints among the variables that rule out certain assignments of values from their domains to them. The problem is to find a solution that satisfies all constraints. (The *constraint optimization problem* is a generalization where every constraint is associated with a non-negative cost and the problem is to find a solution that minimizes the sum of the costs of the unsatisfied constraints.) For example, the NP-hard *map coloring problem* can be modeled as a CSP where the countries are the variables, the set of available colors is the domain of each variable, and the constraints specify that two countries that share a border cannot be colored identically. A solution to a map coloring problem is an assignment of a color to each country. To solve a map coloring problem systematically, one can perform a depth-first search by repeatedly first choosing a country and then a color for it. Different strategies exist for how to choose countries, how to choose their colors, and how to determine when the procedure should undo the last assignment of a color to a country. It should definitely undo the last assignment when two countries that share a border have been colored identically. However, the number of possible map colorings is exponential in the number of countries, which makes such a (systematic) search too slow for a large map. To solve a map coloring problem with hill climbing, one can define the neighbors of a solution to be the solutions that differ in the assignment of a color to exactly one country. The function to be minimized is the number of errors of a solution (*error or loss function*), that is, the number of pairs of countries that share a border but are colored identically. Solving the map coloring problem then corresponds to finding a global minimum of the error function with zero errors, that is, a solution that obeys all coloring constraints. Hill climbing with random restarts often finds a map coloring much faster than search (although this is not guaranteed) but cannot detect that no map coloring exists. Many other NP-hard problems can be solved with hill climbing as well, including the *SATisfiability (SAT) problem*, which is the problem of finding an interpretation that makes a propositional sentence true, or the *Traveling Salesperson Problem* (TSP), which is the problem of finding a shortest route that visits each city of a given set of cities exactly once and returns to the start city.

To avoid getting stuck in local minima, local search cannot only make moves that decrease the function value (*exploiting moves*) but also has to make moves that increase the function value (*exploring moves*). Exploring moves help local search to reach new solutions. We now discuss several versions of local search that decide

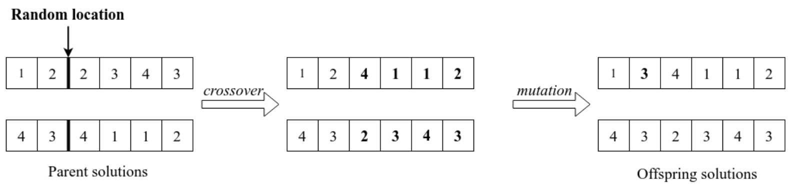


Fig. 1: Illustration of crossover and mutation.

when to make an exploring move, when to make an exploiting move, and which moves to choose (*exploration-exploitation problem*).

Tabu Search

To encourage exploration by avoiding repeated visits of the same solutions, local search can maintain a *tabu list* of previous solutions that it will not move to for a number of steps. *Tabu search* first chooses a random solution and then repeatedly chooses the best neighboring solution that is not in the tabu list and updates the tabu list.

Simulated Annealing

Simulated annealing models the process of heating a metal and then slowly cooling it down to decrease its defects, which minimizes the energy. It first chooses a random solution and then repeatedly chooses a random neighboring solution. If moving to this neighboring solution decreases the function value, it makes the move (exploitation). Otherwise, it makes the move with a probability that is the larger, the less the function value increases and the less time has passed since function optimization started (exploration). Thus, simulated annealing makes fewer and fewer exploring moves over time.

Genetic Algorithms

To allow for the reuse of pieces of solutions in a new context, local search can maintain a set (*population*) of solutions (*individuals* or, synonymously, *phenotypes*) that are represented with strings of a fixed length (*chromosomes* or, synonymously, *genotypes*), rather than only one solution, and then create new solutions from pieces of existing ones. A *Genetic Algorithm* (GA), similar to evolution in biology, first determines the function values of all n solutions in the population and then creates a new population by synthesizing two new solutions $n/2$ times, a process which can easily be parallelized, see Figure 1: 1) It chooses two solutions from the population as parent solutions, each randomly with a probability that is inversely proportional to its function value, and generates two offspring solutions for them. Thus, solutions with small function values (*fitter solutions*) are more likely to reproduce. It splits both parent solutions into two parts by cutting their strings at the same random location. It then reassembles the parts into two new offspring solutions, one of which is the concatenation of the left part of the first parent solution and the right part of the second parent solution and the other one of which is the concatenation of the left part of the second parent solution and the right part of the first parent solution (*crossover*). This *recombination* step ensures that the offspring solutions are genetic mixtures of their parent solutions and can thus be fitter than them (exploitation) or less fit than them (exploration). 2) It changes random characters in the strings of the two offspring solutions. This *mutation* step introduces novelty into the offspring solutions and can thus make them fitter or less fit. Mutation is necessary to create diversity that avoids convergence to a local minimum. For example, if all solutions of a population are identical, then recombination creates only offspring solutions that are identical to their parent solutions. Once all (new) offspring solutions have been generated, they form the next population. This procedure is repeated for many steps (*generations*) until a time bound is reached, the smallest function value of a solution in the new population is sufficiently small, or this function value seems to have converged. Then, the solution in the most recent population with the smallest function value is returned. A good representation of a solution as string is important to ensure that recombination and mutation likely create strings that are solutions and have a chance to increase the fitness. In general, strings are discarded and replaced with new strings if they are not solutions. GAs can move the best solutions from one population to the next one to avoid losing them. To solve a map coloring problem for m countries with a given number of colors, the function to be minimized is the number of pairs of countries that share a border but are colored identically. GAs can represent a solution with a string of m characters, where the ith character represents the assignment of a color to country i. The strings of the initial population are created by assigning colors to countries randomly. Recombination chooses a subset of countries and creates two offspring solutions from two parent solutions, one of which consists of the colors assigned by the first parent solution to the countries in the subset and the colors assigned by the second parent solution to the other countries and the other one of which consists of the colors assigned by the second parent solution to the countries in the subset and the colors assigned by the first parent solution to the other countries. Mutation iterates through the countries and changes the color of each country to a random one with a small probability. Thus, recombination and mutation create strings that are solutions. Actual applications of GAs to map coloring use more complex recombination strategies.

Genetic programming, a form of GAs, uses tree-like representations instead of strings to evolve computer programs that solve given tasks.

Other Discrete Function Minimization Techniques

A discrete function minimization problem under constraints can be formulated like its continuous counterpart (1), except that all or some of the variables have discrete domains. Special cases are *Integer Linear Programming* (ILP) and *Mixed Integer Linear Programming* (MILP), where the objective function and constrained functions are linear in the variables, just like for LP, but all (for ILPs) or some (for MILPs) variables have integer rather than real values. An *LP relaxation*, where all integer-valued variables are replaced with real-valued variables, the resulting LP is solved, and the values of integer-valued variables in the resulting solution are rounded, is often used to find reasonable solutions for ILPs or MILPs quickly. However, since many discrete function minimization problems under constraints can be expressed as ILPs or MILPs, a lot of effort has been devoted to developing more effective but still reasonably efficient solvers for them.

To solve a map coloring problem for m countries with the smallest number of colors under the assumption that there are at most n colors available, one can use the following ILP:

$$\begin{aligned}
\underset{y_1,\ldots,y_m \in \{0,1\}}{\text{minimize}} \quad & \sum_k y_k \\
\text{subject to} \quad & \sum_k x_{ik} = 1, \quad \text{for all } i \in \{1,\ldots,m\}, \\
& x_{ik} + x_{jk} \le 1, \quad \text{for all } i, j \in \{1,\ldots,m\} \text{ and } k \in \{1,\ldots,n\}, \\
& \qquad \text{where countries } i \text{ and } j \text{ share a border}, \\
& x_{ik} \le y_k, \text{ for all } i \in \{1,\ldots,m\} \text{ and } k \in \{1,\ldots,n\}.
\end{aligned}$$

Here, $x_{ik}, y_k \in \{0,1\}$ are variables for all $i \in \{1,\ldots,m\}$ and $k \in \{1,\ldots,n\}$. $x_{ik} = 1$ if and only if country i has color k, and $y_k = 1$ if and only if color k is used. The objective function minimizes the number of colors used. If one only wants to find out whether a map coloring with at most n colors exists, one can use the constant objective function 1. The first constraint states that every country has exactly one color, the second constraint states that countries that share a border do not have the same color, and the third constraint states that a color is used if at least one country has this color.

2.1.3 Example Applications

LPs have been used for controlling oil refinement processes [60]. CSPs have been used for designing complex systems like hybrid passenger ferries [131], mechanical systems like clusters of gear wheels [156], and software services like plan recognition [108]. TSPs have been used for overhauling gas turbine engines, wiring computers, and plotting masks for printed circuit boards [89]. Simulated annealing has been used for global wiring [139] and optimizing designs of integrated circuits [40]. GAs have been used for tuning operating parameters of manufacturing processes [57, 125, 160]. MILPs have been used for scheduling chemical processing systems [36]. Other constraint programming techniques have been used for optimizing the layouts of chemical plants [8].

2.2 Knowledge Representation and Reasoning

Knowledge representation is the task of storing knowledge in a structured way. An *ontology* describes the world in terms of its objects and their properties and relationships. Agents need to represent such knowledge about their environment and themselves in a *knowledge base* with a *knowledge-representation language* that is unambiguous and expressive. They also need to reason with the knowledge. Reasoning is the task of combining existing knowledge, that is, inferring new facts from given ones. For example, knowledge representation is about representing the rules of a coin flip game "heads, I win; tails, you lose" as well as background knowledge about the world (for example, that "heads" and "tails" always have opposite truth values and "I win" and "you lose" always have identical truth values), while reasoning is about inferring that "I win" is guaranteed to hold when playing the game. Facts, like "heads; I win", are expressed with *sentences*. The *syntax* of a knowledge-representation language defines which sentences are well-formed, while the *semantics* of well-formed sentences define their meanings. For example, arithmetic formulas are sentences. "x+2=5" is a well-formed arithmetic formula, and so it makes sense to ask for its meaning, that is, when it is true.

2.2.1 Propositional Logic

We first discuss *propositional logic* as a knowledge-representation language.

Knowledge Representation with Propositional Logic

Sentences in propositional logic represent statements that are either true or false (*propositions*). The syntax specifies how sentences are formed with the *truth values* t(rue) and f(alse), *propositional symbols* (here: in capital letters), and *operators* such as $\neg$, $\wedge$, $\vee$, $\Rightarrow$, and $\Leftrightarrow$, that roughly represent "not", "and", "or" (in the inclusive sense of "and/or"), "if ... then ...", and "if and only if" in English. For example, "heads, I win" can be represented with "H $\Rightarrow$ I", where the propositional symbol H represents "coin flip results in heads" and the propositional symbol I represents "I win". An *interpretation* assigns each propositional symbol a truth value. The semantics specifies for which interpretations a well-formed sentence is true.

Reasoning with Propositional Logic

We want to be able to infer that, whenever the rules of our game are followed, I will win even though we might not know the complete state of the world, such as the outcome of the coin flip. More formally: Whenever an interpretation makes the knowledge base "heads, I win; tails, you lose" (plus the background knowledge) true, then it also makes the sentence "I win" true. In this case, we say that the knowledge base *entails* the sentence. Entailment can be checked by enumerating all interpretations and checking that the definition of entailment holds for all of them.

In our example, it does. However, this procedure is very slow since it needs to check 2^{100} interpretations for knowledge bases that contain 100 propositional symbols. Instead, AI systems use *inference procedures*, like proof by contradiction using the inference rule resolution, that check entailment by manipulating the representations of the knowledge base and the sentence, similar to how we add two numbers by manipulating their representations as sequences of digits.

2.2.2 First-Order Logic

First-Order Logic (FOL) is often more adequate to represent knowledge than propositional logic since propositional logic is often not sufficiently expressive. Sentences in FOL still represent propositions, but FOL is a superset of propositional logic that uses *names* (here: in all caps) to refer to objects, *predicates* (here: in mixed case) to specify properties of objects or relationships among them, *functions* to specify mappings from objects to an object, and *quantification* to express both at least one object makes a given sentence true or that all objects make a given sentence true. For example, the knowledge base “Poodle(FIDO) $\land \forall$ x (Poodle(x) $\Rightarrow$ Dog(x))” expresses that Fido is a poodle and all poodles are dogs. This knowledge base entails the sentence “Dog(FIDO)”. An interpretation now corresponds to functions that map names to objects, predicates to properties or relationships, and functions to mappings from objects to an object. A finite sentence in FOL, different from one in propositional logic, can have an infinite number of interpretations (because one can specify infinitely many objects, such as the non-negative integers, for example, by representing zero and a function that calculates the successor of any given non-negative integer), so we can no longer check entailment by enumerating all interpretations. In general, there is a fundamental limit for FOL since entailment is only *semi-decidable* for FOL, that is, there are inference procedures (such as proof by contradiction using resolution) that can always confirm that a knowledge base entails a sentence, but it is impossible to always confirm that a knowledge base does not entail a sentence (for example, because the inference procedure can run forever in this case).

2.2.3 Other Knowledge Representation Languages

FOL requires sentences to be either true or false. However, the truth of a sentence might need to be quantified, for example, in terms of how true it is (as in “the machine is heavy”), resulting in *fuzzy logic*, or in terms of the likelihood that it is true (as in “the coin flip will result in heads”), resulting in probabilistic reasoning. For example, fuzzy logic allows one to represent vague (but certain) propositions, based on fuzzy set theory where membership in a set (such as the set of heavy machines) is indicated by a degree of membership between zero (the element is not in the set) and one (the element is in the set) rather than the Boolean values false and true. Figure 2 shows membership functions of the fuzzy sets for “light,” “medium,”

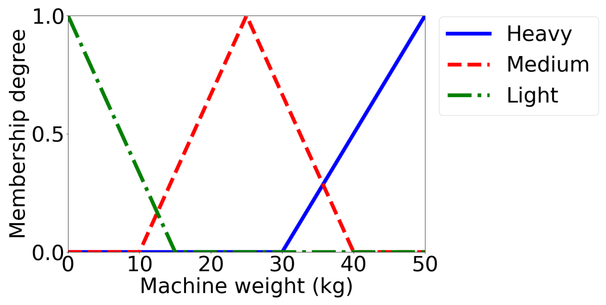


Fig. 2: Membership functions of three fuzzy sets.

and “heavy” machines. Fuzzy control systems are control systems based on fuzzy logic.

In general, FOL has been extended or modified in many directions, for example, to be able to express facts about time (as in “a traffic light, once red, always becomes green eventually”) and to allow for the retraction of entailment conclusions when more facts are added to a knowledge base (*non-monotonic reasoning*). For example, if someone is told that “someone saw a bird yesterday on the roof”, then they will typically assume that the bird could fly (*default reasoning*). But when they are then also told that “the bird had a broken wing”, then they know that it could not fly. Also, proof by contradiction using resolution can be slow for FOL, and its steps can be difficult to explain to users, resulting in hard-to-understand explanations for why a knowledge base entails a sentence. Knowledge-representation languages with less powerful but faster and easier-to-understand reasoning techniques are thus also used, such as rule-based expert systems and semantic networks.

Rule-Based Expert Systems

Rule-based expert systems partition the knowledge base into an “if-then” *rule memory* and a *fact memory* (or, synonymously,*working memory*) since reasoning leaves the rule memory unchanged but adds facts to the fact memory. For example, the rule memory contains “if Poodle(x) then Dog(x)” for our earlier example, and the initial fact memory contains “Poodle(FIDO)”. The inference rule that utilizes that “P(A) $\wedge$ $\forall$ x (P(x) $\Rightarrow$ Q(x))” entails “Q(A)” (*modus ponens*) is used for reasoning. In *forward chaining*, it is used to determine that the knowledge base entails the query “Dog(FIDO)”, which is then added to the fact memory. Forward chaining can be used for configuration planning in design and to find all possible diagnoses. In *backward chaining*, one needs to be given the sentence that one wants to show as being entailed by the knowledge base (*query*), such as “Dog(FIDO)”. The rule “if Poodle(x) then Dog(x)” shows that the knowledge base entails “Dog(FIDO)” if it entails “Poodle(FIDO)”, which it does since “Poodle(FIDO)” is in fact memory. Backward chaining can be used to confirm a given hypothesis in diagnosis. Rule-based expert systems cannot always confirm that a knowledge base entails a sentence.

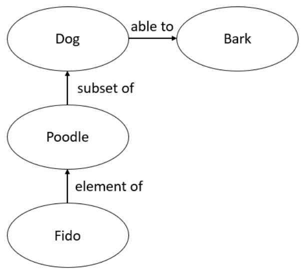


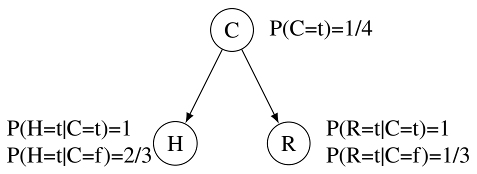
Fig. 3: Semantic network.

For example, they cannot confirm that the knowledge base consisting of the rule memory "Rule 1: if P then R; Rule 2: if ¬P then R" and an empty fact memory entails "R", even though it does. Rule-based expert systems have the advantage of representing knowledge in a modular way and isolating the decisions which rules to apply from the knowledge, making modifications of the knowledge base easy. They have also been extended to logic programming languages, such as Prolog, by allowing commands in the "then" part of rules.

Semantic Networks

Semantic networks, a concept from psychology, are directed graphs that represent concepts visually with nodes and their properties or relationships with directed edges. For example, Figure 3 shows a semantic network that represents the knowledge that Fido is a poodle, all poodles are dogs, and all dogs can bark. Semantic networks can use specialized and fast reasoning procedures of limited reasoning capability that follow edges to reason about the properties and relationships of concepts. This way, they can determine inherited properties. For example, to find out whether Fido can bark, one can start at node "Fido" in the semantic network from Figure 3 and see whether one can reach a node with an outgoing edge that is labeled with "able to" and points to node "Bark" by repeatedly following edges labeled with "element of" and "subset of". They can also disambiguate the meaning of words in sentences. For example, to find out that the word "bank" in the sentence "the bank keeps my money safe" likely refers to the financial institution rather than the land alongside of a body of water, one can determine that node "Money" is fewer hops away in the semantic network that expresses knowledge about the world from the node of the first meaning of bank than the node of the second meaning of bank. Semantic networks can be more expressive than FOL by allowing for default reasoning, but the semantics of edges and reasoning procedures needs to be very carefully defined and some operators (such as "not" and "or") are not easy to represent and reason with.

Bayesian Networks



(a) Bayesian network.

H	R	C	P(H=t ∧ R=t ∧ C=t)
t	t	t	$1 \times 1 \times 1/4 = 1/4$
t	t	f	$2/3 \times 1/3 \times 3/4 = 1/6$
t	f	t	$1 \times 0 \times 1/4 = 0$
t	f	f	$2/3 \times 2/3 \times 3/4 = 1/3$
f	t	t	$0 \times 1 \times 1/4 = 0$
f	t	f	$1/3 \times 1/3 \times 3/4 = 1/12$
f	f	t	$0 \times 0 \times 1/4 = 0$
f	f	f	$1/3 \times 2/3 \times 3/4 = 1/6$

(b) Joint probability table.

H	R	C
t	t	t
t	f	f
t	f	f
f	t	f

(c) Training examples.

Fig. 4: Bayesian network, corresponding joint probability table, and training examples.

Probabilistic reasoning is an especially important extension of FOL because the world is often non-deterministic or the knowledge bases of agents are incomplete, for example, because the agents do not have complete knowledge of the world or do not include known facts in their knowledge bases, for example, to keep the sizes of their knowledge bases manageable. For example, an "if-then" rule "if Jaundice(x) then YellowEyes(x)" states that someone with jaundice always has yellow eyes (in other words, that the *conditional probability* P(YellowEyes(x)=t|Jaundice(x)=t), which is the probability that YellowEyes(x) is true given that Jaundice(x) is true, is 1), which is not true in reality.

Knowledge Representation with Bayesian Networks

For a medical diagnosis system, each symptom or disease is a random variable, and its presence or absence is indicated by the truth value of that random variable. An interpretation assigns a truth value to each random variable. A *joint probability table* maps each interpretation to the probability of the corresponding scenario (*joint probability*). From the joint probability table, one can calculate the conditional probability that a sick person has a certain disease given the presence or absence of some symptoms. However, there are too many interpretations to be able to elicit all joint probabilities from a doctor, and many of the them are too close to zero to result in good estimates. Instead, one can represent the joint probability table with a directed acyclic graph that represents random variables visually with nodes and their conditional dependences with directed edges (*Bayesian network*). Every node is assigned a *conditional probability table* that maps each combination of values of the predecessor nodes of the node to the conditional probability that the node takes on a certain value if the predecessor nodes take on the given values. Figure 4a shows an example Bayesian network with three random variables that take on the values true or false (*Boolean random variables*): C(old), H(igh Temperature), and R(unning Nose). The joint probabilities are calculated as products of conditional probabilities, one from each conditional probability table. Here, P(C=c ∧ H=h ∧ R=r) = P(C=c) P(H=h|C=c) P(R=r|C=c), where the notation means that the equation holds no matter whether C, H, and R are true or false. Thus, the joint probability table in Figure 4b results. The joint probabilities are products of several conditional probabilities. Thus, the joint probabilities are often close to zero, but the conditional probabilities are not necessarily close to zero. Furthermore, the conditional probabilities directly correspond to medical knowledge (such as the probability that a person with a cold has a high temperature), which makes them often easier to estimate. There are 8 possible interpretations, and one thus needs to specify 8-1 = 7 joint probabilities for the joint probability table. The eighth joint probability does not need to be specified since all joint probabilities sum to one. The reason why one only needs to specify five conditional probabilities across all conditional probability tables of the Bayesian network is that the Bayesian network makes conditional independences visible in its graph structure rather than only in its (conditional) probabilities. Two random variables X and Y are *independent* if and only if $P(X{=}x \wedge Y{=}y) = P(X{=}x)\,P(Y{=}y)$. Thus, one needs only two probabilities, namely $P(X{=}t)$ and $P(Y{=}t)$, instead of three probabilities to specify the joint probability table of two independent Boolean random variables. However, making independence assumptions is often too strong. For example, if diseases and symptoms were independent, then the probability that a sick person has a certain disease would not depend on the presence or absence of symptoms. Diseases could thus be diagnosed without seeing the sick person. Making conditional independence assumptions is often more realistic. Two random variables X and Y are *conditionally independent* given a third random variable Z if and only if $P(X{=}x \wedge Y{=}y|Z{=}z) = P(X{=}x|Z{=}z)\,P(Y{=}y|Z{=}z)$, in other words, X and Y are independent if one knows the value of Z (no matter what that value is). The graph structure of a Bayesian network implies conditional independences that hold no matter what the values of the conditional probabilities in its conditional probability tables are. Here, $P(H{=}h \wedge R{=}r|C{=}c) = P(H{=}h|C{=}c)\,P(R{=}r|C{=}c)$. While a Bayesian network corresponds to exactly one joint probability table (meaning that both of them specify the same joint probabilities), a joint probability table can correspond to many Bayesian networks (namely, one for each factorization of the joint probabilities, such as $P(C{=}c \wedge H{=}h \wedge R{=}r) = P(H{=}h \mid R{=}r \wedge C{=}c)\,P(R{=}r \mid C{=}c)\,P(C{=}c)$ or $P(C{=}c \wedge H{=}h \wedge R{=}r) = P(C{=}c \mid H{=}h \wedge R{=}r)\,P(H{=}h \mid R{=}r)\,P(R{=}r)$) – and these Bayesian networks can differ in their number of conditional probabilities after their simplification. A Bayesian network with few conditional probabilities can typically be found by making most of its edges go from causes to effects, for example, from diseases to symptoms (as in our example). Then, one typically needs to specify many fewer conditional probabilities than joint probabilities and speeds up reasoning.

Reasoning with Bayesian Networks

Reasoning with Bayesian networks means calculating probabilities, often conditional probabilities that one random variable takes on a given value if the values of some other random variables are known, for example, the probability $P(C{=}t|R{=}f)$ that a sick person with no running nose has a cold or the probability $P(C{=}t|H{=}f \wedge R{=}f)$ that a sick person with normal temperature and no running nose has a cold. In general, reasoning is NP-hard but can often be scaled to large Bayesian networks by exploiting their graph structure in form of their conditional independence assumptions.

2.2.4 Example Applications

FOL has been used for verifying software [119] and proving theorems [35]. Fuzzy logic has been used for controlling machines, including maintaining a constant feed rate for weight belt feeders [157], increasing the machining efficiency for rough milling operations [46], and improving vehicle control of anti-lock braking systems [91, 158]. Rule-based expert systems have been used for tele-monitoring heart failures [120], analyzing satellite images [124], and developing load-shedding schemes for industrial electrical plants [25]. Semantic networks have been used for understanding consumer judgments [45, 41] and capturing knowledge of production scheduling [116]. Bayesian networks have been used in manufacturing for diagnosing and predicting faults [20, 143], calculating failure rates for maintenance planning [59], and predicting the energy consumption of manufacturing processes [98].

2.3 Planning

Planning is the optimization task of using knowledge to determine how to behave to maximize task performance. It can use the generic optimization techniques discussed earlier but often solves the specific optimization problem of finding an (ideally) shortest or cost-minimal sequence of actions that allows an agent to transition from a given start state to a given goal state and thus uses specialized optimization techniques to exploit the structure of this problem well and run fast. A *state* characterizes the information that an agent needs to have about the past and present to choose actions in the future that maximize its task performance. For example, a soda machine does not need to remember in which order a customer has inserted which coins. It only needs to remember how much money the customer has already inserted. An important part of planning is *scheduling*, which is the assignment of resources to plans (including when actions should be executed). We first discuss planning for an agent in the absence of other agents (*single-agent systems*) and then planning in the presence of other agents (*multi-agent systems*).

2.3.1 Deterministic Planning in Single-Agent Systems

Deterministic planning problems assume that the execution of the same action in the same state always results in the same successor state and the actions thus have deterministic effects. The input of a deterministic planner is the start state of an agent (which is typically its current state), the (desired) goal state, and a set of actions that the agent can execute to transition from one state to another one. Its output is a *plan*,

1	2	3
4	5	6
	7	8

Start state

1	2	3
4	5	6
7	8	

Goal state

```
Action MoveTile(x,y,z) - "Move tile x from location y to location z"
Precondition set = {At(x,y), MissingTile(z), Adjacent(y,z)}
Add effect set = {At(x,z), MissingTile(y)}
Delete effect set = {At(x,y), MissingTile(z)}

Start state = {At(T1, L1), At(T2, L2), At(T3, L3), At(T4, L4), At(T5, L5),
               At(T6, L6), At(T7, L8), At(T8, L9), MissingTile(L7),
               Adjacent(L1, L2), Adjacent(L2, L3), Adjacent(L4, L5),
               Adjacent(L5, L6), Adjacent(L7, L8), Adjacent(L8, L9),
               Adjacent(L1, L4), Adjacent(L4, L7), Adjacent(L2, L5),
               Adjacent(L5, L8), Adjacent(L3, L6), Adjacent(L6, L9),
               Adjacent(L2, L1), Adjacent(L3, L2), Adjacent(L5, L4),
               Adjacent(L6, L5), Adjacent(L8, L7), Adjacent(L9, L8),
               Adjacent(L4, L1), Adjacent(L7, L4), Adjacent(L5, L2),
               Adjacent(L8, L5), Adjacent(L6, L3), Adjacent(L9, L6)}

Goal state  = {At(T1, L1), At(T2, L2), At(T3, L3), At(T4, L4), At(T5, L5),
               At(T6, L6), At(T7, L7), At(T8, L8), MissingTile(L9)}
```

Fig. 5: Eight puzzle instance.

which is an (ideally) shortest or, if costs are associated with the actions, cost-minimal sequence of actions that the agent can execute to transition from the start state to the goal state. Since many planning problems have more than 10^{100} states [112], one often trades off solution quality for runtime and is satisfied with a short sequence of actions. Consider, for example, the eight puzzle, which is a toy with eight numbered square tiles in a square frame. A state specifies which tiles are in which locations (configuration of the eight puzzle). The current state can be changed by repeatedly moving a tile that is adjacent to the location with the missing tile to that location. Figure 5 shows a start state in the upper-left corner and a goal state in the lower-left corner. A shortest plan is to move the 7 tile to the left with `MoveTile(T7,L8,L7)` and then the 8 tile to the left with `MoveTile(T8,L9,L8)`.

Specifying Deterministic Planning Instances with STRIPS

Deterministic planning is essentially the graph-search problem of finding a short path from the start state to the goal state on the directed graph (*state space*) that represents states with vertices and actions with edges. However, the state space of the eight puzzle has 9!/2 = 181,440 states (since, to populate the eight puzzle, the 1 tile can be placed in any of the 9 locations, the 2 tile can be placed in any of the remaining 8 locations, and so on, but only half of the states are reachable from any given state), which makes the state space tedious to specify explicitly by enumeration. Even larger state spaces, such as those with more than 10^{100} states, cannot be specified explicitly by enumeration since both the necessary time and amount of memory are too large. Therefore, one specifies state spaces implicitly with a formal specification language for planning instances, such as STRIPS. STRIPS was first used for the *Stanford Research Institute Problem Solver*, an early planner, which explains its name. States are specified with a simplified version of FOL as conjunctions (often written as sets) of predicates whose arguments are names (*grounded predicates*). Actions are specified as parameterized *action schemata*, that define when an action can be executed in a state (namely, when one can replace

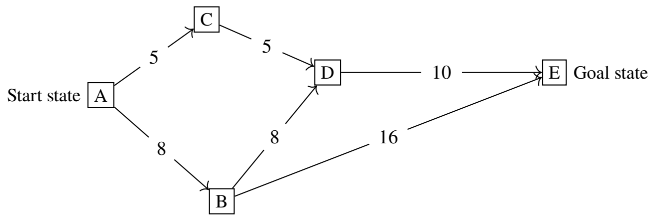


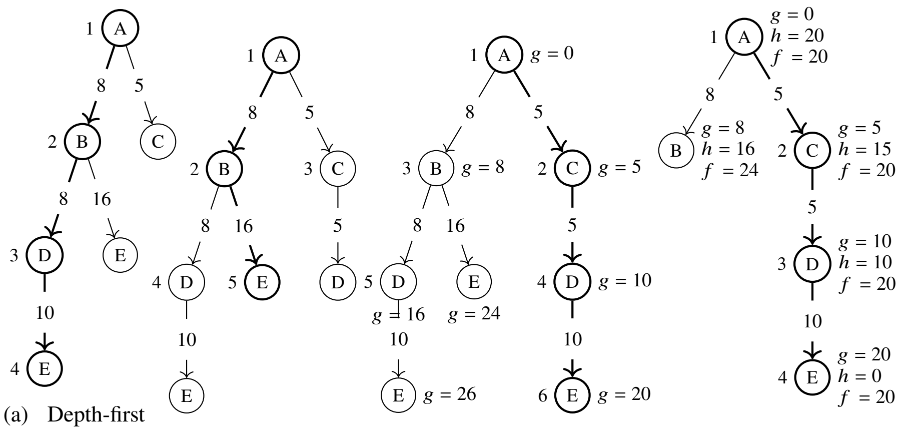
Fig. 6: A path-finding instance.

each parameter with a name so that all predicates in its *precondition set* are part of the state) and how to calculate the resulting successor state (namely, by deleting all predicates in its *delete effect set* from the state and then adding all predicates in its *add effect set*). Figure 5 shows the input of the planner for our example. The *Planning Domain Dejinition Language* (PDDL) [39] is a more expressive version of STRIPS that includes typing, negative preconditions, conditional add and delete effects, and quantification in preconditions, add effects, and delete effects. Its more advanced version PDDL2.1 [37] further supports optimization metrics, durative (rather than instantaneous) actions, and numeric preconditions, add effects, and delete effects. PDDL3 [38] further supports constraints over possible actions in the plan and the states reached by them (*trajectory constraints*) as well as soft trajectory constraints, that are desirable but do not necessarily have to be satisfied (*preferences*).

Solving Deterministic Planning Instances

Deterministic planning instances can be solved by translating them into different optimization instances, such as (M)ILP instances [12, 105] or SAT instances [64, 110]. Deterministic planning instances can also be solved with search techniques that find short paths in the given state space. These search techniques build a *search tree* that represents states with nodes and actions (that the agent can execute to transition from one state to another) with edges. Initially, the search tree contains only the root node labeled with the start state. In each iteration, the search technique selects an unexpanded leaf node of the search tree for expansion. If no such leaf node exists, then it terminates and returns that no path exists. Otherwise, if the state of the selected node is the goal state, then it terminates and returns the action sequence of the unique path in the search tree from the root node to the selected node. Otherwise, it *expands* the selected node as follows: For each action that can be executed in the state of the selected node, it creates an edge that connects the selected node with a new node labeled with the successor state reached when the agent executes the action in the state of the selected node. Finally, it repeats the procedure.

Different search techniques differ in the information that they use for selecting an unexpanded fringe node. All of the following search techniques find a path from the start state to the goal state or, if none exists, report this fact (they are *complete*) when used on state spaces with a finite number of states and actions. Figure 6 shows a path-finding instance, and Figure 7 shows the resulting search trees.



(a) Depth-first search. (b) Breadth-first search. (c) Uniform-cost search. (d) A* search.

Fig. 7: Search trees for depth-first search, breadth-first search, uniform-cost search, and A* search for the path-finding instance from Figure 6. The admissible h-value of a node for the A* search is the goal distance of its state. The numbers indicate the order of node expansions, and the found paths are shown in bold. Not used here is the common optimization to prune nodes once a node labeled with the same state has been expanded.

Uninformed (or, synonymously, *blind*) *search* techniques use only information from the search tree for selecting an unexpanded fringe node. The *g-value* of a node n is the sum of the edge costs from the root node to node n. *Depth-first search* always selects an unexpanded fringe node with the largest g-value under the assumption that all edge costs are one (no matter what the actual edge costs are), that is, that the g-value of a node n is the number of edges from the root node to node n. To be complete, it has to prune nodes if one of their ancestors is labeled with the same state, which avoids paths with cycles. It is not guaranteed to find shortest paths (even if all actual edge costs are one) but can be implemented with a stack, resulting in a memory consumption that is linear in the search depth. *Uniform-cost search* always selects an unexpanded fringe node with the smallest g-value (using the actual edge costs). It is guaranteed to find shortest paths and can be implemented with a priority queue. *Breadth-first search* always selects an unexpanded fringe node with the smallest g-value under the assumption that all edge costs are one (no matter what the actual edge costs are). It is a special case of uniform-cost search since uniform-cost search behaves like breadth-first search if all edge costs are one. It is guaranteed to find shortest paths only if all actual edge costs are one and can be implemented with a first-in first-out queue. Both uniform-cost search and breadth-first search can result in a memory consumption that is exponential in the search depth. *Iterative deepening* implements a breadth-first search with a series of depth-first searches of increasing search depths. It is guaranteed to find shortest paths when breadth-first search does, can be implemented with a stack, and results in a memory consumption that is linear in the search depth but has a runtime overhead compared to breadth-first search. More sophisticated linear-memory search techniques also exist, including in case all edge costs are not one.

Informed (or, synonymously, *heuristic*) *search* techniques use additional problem-specific information for selecting an unexpanded fringe node. The *h-value* of a node *n* is an estimate of the goal distance of the state of node *n*, which is the smallest sum of the edge costs from the state of node *n* to the goal state. An h-value is *admissible* if and only if it is not an overestimate of the corresponding goal distance. The *f-value* of a node is the sum of its g- and h-values. *A** always selects an unexpanded fringe node with the smallest f-value, that is, the smallest estimated sum of the edge costs from the start state via the state of the node to the goal state. Uniform-cost search is a special case of A* since A* behaves like uniform-cost search if all h-values of A* are zero. A* is guaranteed to find shortest paths (if all h-values are admissible) and can be implemented with a priority queue. There exist linear-memory search variants of A*. The h-values being admissible guarantees that the f-values of all nodes are not overestimates, a principle known in AI as "*optimism in the face of uncertainty*" (or missing knowledge). This way, when A* expands a node, the state of the node is either on a shortest path from the start state to the goal state (which is good) or not (which allows A* to discover the fact that the state is not on a shortest path). On the other hand, if some h-values are not admissible, then A* might not expand all nodes on a shortest path (because their f-values are too large) and is thus not guaranteed to find shortest paths. An admissible h-value of node *n* is typically found as the smallest sum of the edge costs from the state of node *n* to the goal state in a state space that contains additional actions (*relaxation* of the original state space) because adding actions cannot increase the goal distances. For example, for path finding on a map of cities and their highway connections, one could add actions that move off-road on a straight line from any city to any other city. The resulting admissible h-value of a node is the straight-line distance from the city of the node to the goal city. For the eight puzzle, one could add actions that move a tile on top of a neighboring tile. The resulting admissible h-value of a node is the sum, over all tiles, of the differences in the x- and y-coordinates of the location of a tile in the configuration of the node and its location in the goal configuration. For a planning problem specified in STRIPS, one needs large admissible h-values to keep the number of expanded nodes manageable due to the large state space since the larger the admissible h-values, the fewer nodes A* expands. One could delete some or all of the elements of the precondition sets of action schemata but often deletes all elements of their delete sets instead. In both cases, additional actions can be executed in some states because the actions need to satisfy fewer preconditions in the first case and because previous actions did not delete some of their preconditions in the second case.

So far, we have assumed a *forward search* from the start state to the goal state. For this, one needs to determine all possible successor states of a given state. One can also perform a *backward search* from the goal state to the start state, provided that one is able to determine all possible predecessor states of a given state. If both forward and backward searches are possible, then one should choose the search direction that results in the smaller average number of child nodes of nodes in the search tree (*average branching factor*). *Bi-directional search* searches simultaneously in both directions to search faster and with less memory. If the state space is specified with STRIPS, then it is easy to determine all possible successor states of a given state and thus implement a forward search, while it is more difficult to determine all possible predecessor states of a given state.

2.3.2 Probabilistic Planning in Single-Agent Systems

Actions often do not have deterministic effects in practice. For example, a sick person might not respond to their medicine. The resulting *probabilistic* (or, synonymously, *decision-theoretic*) *planning* problems are often specified with *(totally observable) Markov Decision Processes* (MDPs) or *Probabilistic PDDL* (PPDDL) [151]. MDPs assume that 1) the current state is always known (for example, can be observed after every action execution), and the probability distributions over the cost and the successor state resulting from an action execution 2) are known and 3) depend only on the executed action and the state it is executed in (*Markov property*). For deterministic planning problems, a plan is a sequence of actions. But, for MDPs, a plan needs to specify which action to execute in each state that can be reached during its execution. It is a fundamental result that a plan that minimizes the expected total (discounted) plan-execution cost can be specified as a function that assigns each state the action that should be executed in that state (*policy*). (The *discount factor* weighs future costs less than current ones and is primarily used as a mathematical convenience since it ensures that all infinite sums are finite.) Policies that minimize the expected total (discounted) plan-execution cost can be found with stochastic dynamic programming techniques, such as value iteration and policy iteration.

Assumptions 1-3 from above do not always hold in practice. *Partially Observable MDPs* (POMDPs) relax Assumption 1 by assuming that the current state can be observed after every action execution only with noisy observations, which means that only a probability distribution over the current state is known. For example, the disease of a sick person is not directly observable, but the observed symptoms provide noisy clues about it. A policy now assigns each probability distribution over the current state the action that should be executed in this belief state, which results in more complicated and slower stochastic dynamic programming techniques than for MDPs since the number of states is typically finite but the number of belief states is infinite.

2.3.3 Planning in Multi-Agent Systems

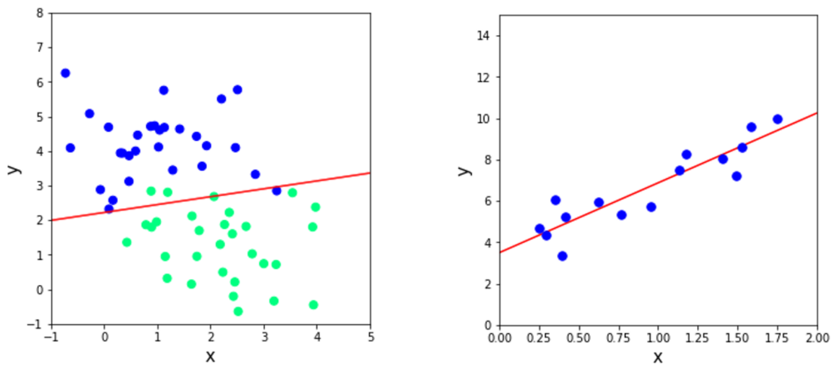
Multi-agent systems have become more important as parts of companies have been connected with the internet, computational devices have been increasingly networked as part of the internet of things, and teams of robots have been fielded successfully. Several agents can be more robust than a single agent since they can compensate for the failure of some agents. Several agents can also act in parallel and reduce the task-completion time. Centralized planning techniques for multi-agent systems can be obtained from planning techniques for single-agent systems and maximize the performance of the team (*social welfare*) but might raise privacy concerns for the involved agents. Decentralized (and distributed) planning techniques correspond to the many ways of making decentralized collective decisions in multi-agent systems, for example, with negotiating, voting, or bidding. *Competitive agents* are self-interested, that is, maximize their own performance, while *cooperative agents* maximize the performance of the team, for example, because they are bound by social norms or contracts (in case of humans) or are programmed this way (in case of robots). In both cases, one needs to understand how the agents can interact (*mechanism*) and what solutions can result from these interactions. One often wants to design the mechanism so that the solutions have desirable properties. For this, AI can utilize insights from other disciplines. Economics has mostly studied competitive agents, for example, in the context of non-cooperative game theory or auctions. An important part of the design of an auction mechanism in this context is to ensure that agents cannot game the system, for example, with collusion or shilling. Operations research, on the other hand, has mostly studied cooperative agents, for example, in the context of vehicle routing. Concepts from economics also apply to cooperative agents since they can, for example, use cooperative auctions to assign tasks to themselves: Each agent bids on all tasks and is then assigned the tasks that it must execute, for example, the tasks for which it was the highest bidder. An important part of the design of an auction mechanism in this context is to design the auction so that, if each agent maximizes its own performance, the team maximizes social welfare. Different from many applications in economics, it is often also important that task allocation is fast.

2.3.4 Example Applications

PDDL has been used for managing greenhouses [49] and composing and verifying web services [102, 54]. A* has been used for planning the paths of automated guided vehicles [142]. MDPs and POMDPs have been used for planning robot motions [133], tuning the parameters of wireless sensor networks [67, 96], and managing power plants [109]. Planning for multi-agent systems has been used for planning multi-view drone cinematography [97], production planning for auto engine plants [101], and assembling furniture with robots [69].

2.4 Machine Learning

Machine learning is the task of acquiring knowledge from observations of the world, often because knowledge is hard to specify by hand. Machine learning typically involves specifying a set of *considered functions* as possible models of some aspect of the world and choosing a good function from this set based on the observations. The set of considered functions is often specified in form of a parameterized function



(a) Classification instance with training examples (shown as small blue and green dots) with two real-valued features x and y and a label with the discrete values blue or green. The predicted values of the label on one side of the red line are blue, and the predicted values on the other side are green. Not all predictions are accurate.

(b) Regression instance with training examples (shown as small blue dots) with one real-valued feature x and real-valued label y. The red line shows the predicted values of the label for all possible values of x. Not all predictions are accurate.

Fig. 8: Illustration of classification and regression.

and using optimization techniques to determine its parameters. The key challenge is to ensure that the resulting learned function generalizes beyond the observations. For example, it is difficult to program an agent to ride a bicycle without falling down, and it thus makes sense to let it learn from experience (that is, past observations) how to do it well, which involves handling situations that the agent did not experience during learning.

We discuss three classes of machine learning techniques. Assume that one wants to learn which actions to execute in different situations. Supervised learning is similar to a teacher presenting examples for how to behave optimally in many situations, unsupervised learning is similar to experiencing many situations and trying to make sense of them without any further information, and reinforcement learning is similar to experiencing many situations, experimenting with how to behave in each of them, and receiving feedback as to how good the behavior was (for example, when falling down while learning how to ride a bicycle).

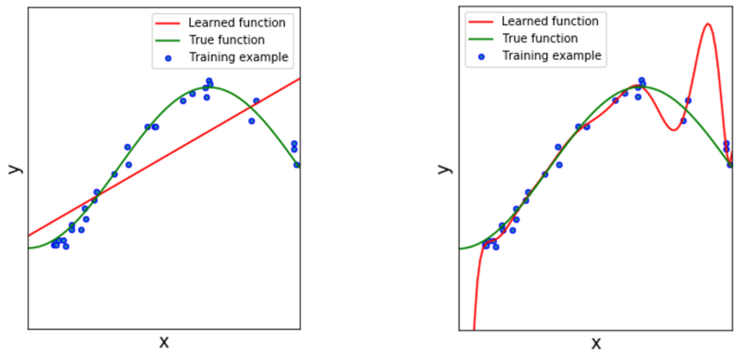
2.4.1 Supervised Learning

In *supervised learning*, an agent is provided with observations in form of *labeled examples*, where the agent knows the values of a fixed number of features and the value of the label of each example. Its task is to predict the values of the label of the encountered *unlabeled examples* in use, where the agent knows only the values of the features. This task can be framed as learning a function that maps examples to the predicted values of the label. There is a possibly infinite set of considered functions (*hypothesis space*), and one wants to find the best function from this set, which is ideally the *true function* that actually generates the values of the label. If the set of possible values of the label is discrete, then this is a *classification task*, see Figure 8a. Otherwise, it is a *regression task*, see Figure 8b. For example, recognizing handwritten digits from the pixels in images or spam emails from the number of times certain words occur in them are classification tasks, while predicting the life expectancy of a person from their medical history is a regression task.

An objective function captures how well a function predicts the values of the label of examples encountered in use (*error function* or, synonymously, *loss function*). The task is to choose the function from the set of considered functions that minimizes the error function. Examples of error functions are mappings from the set of considered functions to the resulting number of prediction errors for classification tasks and the resulting average of the squared differences between the true and predicted values of the label (*mean squared error*) for regression tasks. One often characterizes the number of prediction errors for classification tasks using precision and recall. Assume, for example, that 100 emails are provided to a spam email detection system, of which 20 are spam emails (*actual positive examples*). If the system identifies 15 emails as spam emails (*identified positive examples*) and 10 of them are indeed spam emails (*correctly identified positive examples*), then the number of correctly identified positive examples divided by the number of identified positive examples (*precision*) is 10/15 and the number of correctly identified positive examples divided by the number of actual positive examples (*recall*) is 10/20.

Since there are too many examples that could be encountered in use, one cannot provide all of them as labeled examples. Machine learning thus needs to generalize from the labeled examples to the ones encountered in use. To make this possible, one typically assumes that the labeled examples are independently drawn from the same probability distribution as the examples encountered in use, resulting in them being *independent and identically distributed* (iid). The labeled examples serve different purposes and are therefore often partitioned randomly into three sets, namely training examples, validation examples, and (if needed) test examples:

- The *training examples* are used to learn the function with a given machine learning technique and its parameters. For example, one might want to find the polynomial function of a given degree that has the smallest mean squared error on the examples encountered in use. Since these examples are not available during learning, one settles for finding the polynomial function of the given degree that has the smallest mean squared error on the training examples, which can be found, for example, with gradient descent. If the error is small, then one says that the function *fits* the training examples well. Any machine learning technique has to address underfitting and overfitting, see Figure 9. *Underfitting* means that the learned function is not similar to the true function, typically because the true function is not in the set of considered functions (resulting in high *bias*). For example, if the set of considered functions contains only linear functions for regression tasks (*linear regression tasks*) but the true function is a polynomial function of higher degree, then the best linear function is likely not similar to



(a) The learned function underfits. (b) The learned function overfits.

Fig. 9: Illustration of underfitting and overfitting.

the true function and thus fits neither the training examples nor the examples encountered in use well, as shown in Figure 9a. *Overfitting* means that the learned function fits the training examples well but has adapted so much to the training examples that it is no longer similar to the true function even if the true function is in the set of considered functions, typically because there is lots of sampling noise in the training examples due to their small number or because the error function or machine learning technique is too sensitive to the noise in the feature or label values of the training examples (resulting in high *variance*). For example, if the training examples consist of a single photo that contains a cat, then a function that predicts that every photo contains a cat fits the training examples well but is likely not similar to the true function and thus likely does not fit the examples encountered in use well. Similarly, if there is noise in the values of the features or the label for regression tasks, then a polynomial function of higher degree than the true function can fit the training examples better than the true function but is likely not similar to the true function and thus likely does not fit the the examples encountered in use well, as shown in Figure 9.
The *bias-variance dilemma* states that there is a trade-off between the bias and the variance. To balance them, one can enlarge or reduce the set of considered functions, for example, by increasing or decreasing the number of parameters to be learned. Other techniques exist as well. For example, *Least Absolute Shrinkage and Selection Operator* (LASSO) regression is a popular technique to reduce overfitting for linear regression tasks. It minimizes the mean squared error plus a weighted penalty term (*regularization term*) that is the L1 norm of the weights of the linear function to be learned.

- The *validation examples* are used to select parameters for the machine learning technique (*hyperparameters*) that allow it to learn a good function, which has to be done with examples that are different from the training examples (*cross validation*). Hyperparameters are, for example, the degree of the polynomial function

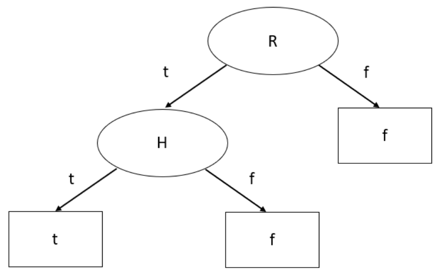


Fig. 10: Decision tree for the example from Figure 4c.

to be learned for polynomial regression tasks or the weight of the regularization term for LASSO. They can be selected manually or automatically with exhaustive search, random search, or machine learning techniques.

- The *test examples* are used as proxy for the examples encountered in use when assessing how well the learned function will likely do in use.

The more training and validation examples are available, the better a function can be learned. A fixed split of the non-test examples into training and validation examples does not make good use of the non-test examples, which is a problem if their number is small. One therefore often uses each non-test example sometimes as training example and sometimes as validation example. For example, *k-fold cross validation* partitions the non-test examples into k sets and then performs k rounds of learning for given hyperparameters, each time using the examples in one of the partitions as validation examples and the remaining non-test examples as training examples. After each round, it calculates how well the resulting function predicts the values of the label of the validation examples. The hyperparameters are then evaluated according to the average of these numbers over all k rounds.

Supervised machine learning techniques differ in how they represent the set of considered functions. We discuss several examples in the following because none of them is universally better than the others (*no free lunch theorem*) [146]. We do not discuss variants of supervised learning, such as the one where one synthesizes examples and then is provided with their values of the label at a cost (*active learning*) or the one where one uses the knowledge gained from solving one classification or regression task to solve a similar task with fewer examples or less runtime (*transfer learning*), which is important since machine learning techniques are typically both training data and runtime intensive. *Meta-learning*, which is learning from the output of other machine learning techniques, can be used to improve the learning ability of machine learning techniques by learning to learn, for example, across related classification or regression tasks for multi-task learning [75].

Decision Tree Learning

Decision tree learning represents the set of considered functions with trees where each non-leaf node is labeled with a feature, the branches of each non-leaf node are labeled with a partition of the possible values of that feature, and each leaf node is labeled with a value of the label (*decision trees*), see Figure 10. Decision trees can be used for both classification and regression tasks (*classification and regression trees*). The value of the label of an unlabeled example is determined by starting at the root node and always following the branch that is labeled with the value of that feature for the example that labels the current node. The label of the leaf node eventually reached is the predicted value of the label for the example. For example, the decision tree in Figure 10 predicts that a sick person with normal temperature and no running nose does not have a cold. Decision tree learning constructs a small decision tree, typically from the root node down. It chooses the most informative feature for the root node (often using metrics from information theory) and then recursively constructs each subtree below the root node. Overfitting can be reduced by limiting the depth of a decision tree or, alternatively, pruning it after its construction. It can also be reduced by using *ensemble learning* to learn several decision trees (*random forest*) and then letting them vote on the value of the label by outputting the most common value among them for classification tasks and the average of their values for regression tasks. One can also use meta-learning to determine how to combine the predictions of the decision trees.

Naïve Bayesian Learning

Naïve Bayesian learning represents the set of considered functions with Bayesian networks of a specific graph structure. The graph structure of the Bayesian networks is the one given in Figure 4a, with an edge from the class to each feature. Naïve Bayesian learning estimates the conditional probabilities, typically using frequencies. Assume, for example, a diagnosis task where the features are the symptoms H(igh Temperature) and R(unning Nose) in Figure 4c and the label is the disease C(old). Three training examples have C=f, and two of those have H=t, so P(H=t|C=f) = 2/3. Overall, the conditional probabilities in Figure 4a result for the given training examples. Bayesian networks of the simple graph structure shown in Figure 4a are called *naïve Bayesian networks* because the conditional independences implied by their graph structure might not be correct. In fact, the conditional independence $P(H=t \wedge R=t|C=f) = P(H=t|C=f)\ P(R=t|C=f)$ does not hold for the training examples since $P(H=t \wedge R=t|C=f) = 0 \neq 2/3\ 1/3 = P(H=t|C=f)\ P(R=t|C=f)$. Thus, naïve Bayesian learning can underfit but is often expressive enough for the learned function to be sufficiently similar to the true function and then has the advantage that it reduces the set of considered functions compared to Bayesian networks with more complex graph structures and thus reduces overfitting. When one needs to predict the value of the label of an unlabeled example, for example, H=f and R=f, one calculates $P(C=t|H=f \wedge R=f) = P(C=t \wedge H=f \wedge R=f)\ /\ P(H=f \wedge R=f) = P(C=t \wedge H=f \wedge R=f)\ /\ (P(C=t \wedge H=f \wedge R=f) + P(C=f \wedge H=f \wedge R=f)) = P(C=t)\ P(H=f|C=t)\ P(R=f|C=f)\ /\ (P(C=t)\ P(H=f|C=t)\ P(R=f|C=f) + P(C=f)\ P(H=f|C=f)\ P(R=f|C=f)) = 0$ to determine the probability that the value of the label is true, that is, that a sick person with normal

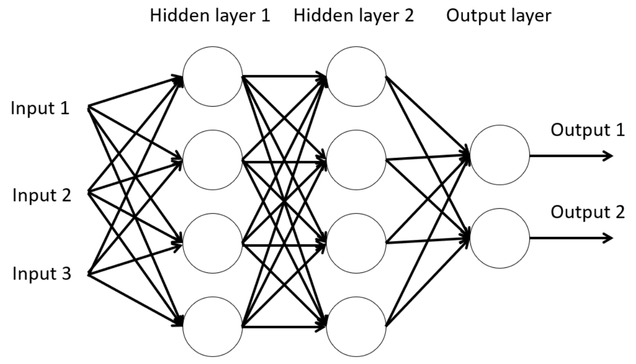
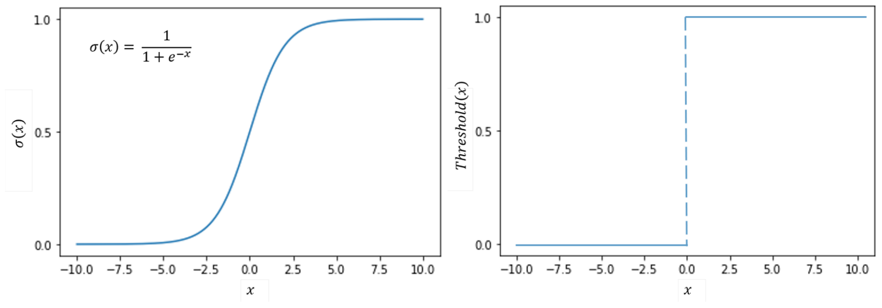


Fig. 11: NN with three inputs, fully connected layers, and two outputs.



(a) Sigmoid function.

(b) Threshold function.

Fig. 12: Sigmoid function and its corresponding threshold function.

temperature and no running nose has a cold. This calculation is essentially the one of *Bayes' rule* P(C=t|H=f) = P(C=t ∧ H=f) / P(H=f) = P(C=t ∧ H=f) / (P(C=t ∧ H=f) + P(C=f ∧ H=f)) = P(C=t) P(H=f|C=t) / (P(C=t) P(H=f|C=t) + P(C=f) P(H=f|C=f)) but generalized to multiple observed symptoms with the conditional independences implied by the graph structure of naïve Bayesian networks.

Neural Network Learning

Neural network learning represents the set of considered functions with artificial *Neural Networks* (NN), see Figure 11. NNs are directed acyclic graphs, inspired by the networks of neurons in brains, that represent primitive processing units (*perceptrons*) with nodes and the flow of information from one perceptron to another with directed edges. A perceptron has a number of real-valued inputs and produces one real-valued output. If a single perceptron is used for learning, then the inputs of the perceptron are the feature values and its output is the value of the label. A weight is associated with each input. The perceptron calculates the weighted sum of its inputs and then applies a nonlinear *activation function* to the weighted sum to create its output. The activation function is typically monotonically nondecreasing. An example is the *sigmoid function* $\sigma(x) = 1/(1 + e^{-x})$, see Figure 12, which is a differentiable approximation of a threshold function to facilitate gradient descent. Perceptron learning finds the weights (including the amount of horizontal translation of the activation function, which is often expressed as weight) that minimize the error function. A single perceptron can essentially represent all functions from n real-valued feature values (for any n) to a label with two values where all examples that the function maps to one value of the label lie on one side of an $(n - 1)$-dimensional separating plane and all examples that the function maps to the other value lie on the opposite side of the plane (the examples are *linearly separable*). This might be expressive enough for the learned function to be sufficiently similar to the true function and then has the advantage that it reduces the set of considered functions compared to a NN and thus reduces overfitting.

Often, however, a single perceptron is not expressive enough (for example, because it cannot even specify an exclusive or function), which is why one connects perceptrons into a NN. If a NN is used for learning, then the inputs of a perceptron in the NN can be either feature values, which are the inputs of the NN, or the outputs of other perceptrons. The output of a perceptron in the NN can be an input to multiple other perceptrons and/or the value of the label, which is the output of the NN. The perceptrons are often organized in layers, see Figure 11. NNs can have multiple outputs (for example, one for each value of the label, such as the possible diseases of a sick person), which are produced by the *output layer*. Sometimes they are postprocessed by a *softmax layer* to yield a probability distribution over the possible values of the label. The remaining layers are the *hidden layers*. A NN is *deep* if and only if it has many hidden layers. There is typically a regularity in the connections from layer to layer. For example, in *fully connected layers*, the output of each perceptron in a layer is the input of all perceptrons in the next layer, see Figure 11. NN learning often, but not always, takes the graph structure of the NN as given and uses gradient descent to find the weights of all perceptrons of the NN that minimize the error function. For example, the *backpropagation* technique adjusts the weights repeatedly with one forward pass through the layers (that calculates the outputs of all perceptrons for a given training example) and one backward pass (that calculates the gradient of the error function with respect to each weight). Specialized NNs exist for specific tasks:

- *Convolutional NNs* (CNNs) are NNs that are specialized for processing matrix-like data, such as images. They use several kinds of layers in addition to the layers already described. For example, a *convolutional layer* (without kernel flipping) is associated with a filter whose values are learned. The filter is a matrix A (*kernel*) that moves over the matrix B of the outputs of the previous layer with a step size that is given by the *stride*. It calculates the dot product of matrix A and the submatrix of matrix B that corresponds to each location. The matrix of these dot products is the output of the convolutional layer, see Figure 13. Overall, a convolutional layer extracts local features. A *pooling layer* is similar to a convolutional layer except that, instead of calculating the dot product, it returns

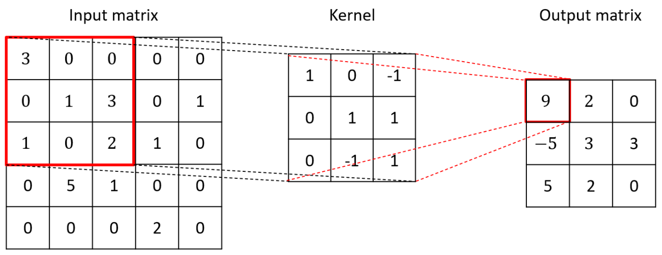


Fig. 13: Illustration of convolution (without kernel flipping) with stride one.

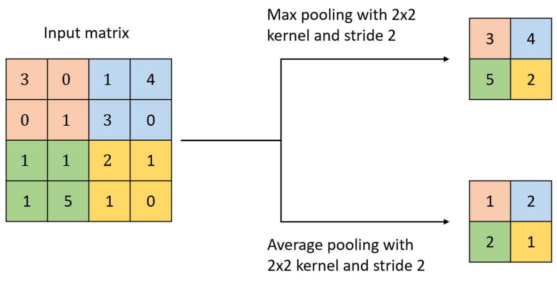

Input matrix
3 0 1 4
0 1 3 0
1 1 2 1
1 5 1 0
Max pooling with 2x2 kernel and stride 2
3 4
5 2
Average pooling with 2x2 kernel and stride 2
1 2
2 1

Fig. 14: Illustration of max pooling and average pooling with stride two.

the maximum value (*max pooling*) or the average value (*average pooling*) of the elements of the submatrix of matrix B that corresponds to each location, see Figure 14. Overall, a pooling layer compresses the information in a lossy way.

- *Recurrent NNs* (RNNs) are NNs that are specialized for classifying sequences (of examples) over time, including handwriting and speech. They use internal state to remember the outputs of some perceptrons from the previous time step and can use them as input of some perceptrons in the current time step. *Long Short-Term Memory* (LSTM) networks are versions of RNNs that make it easier to maintain the internal state by using three gates to the memory, namely one that decides what to store in memory (*input gate*), one that decides what to delete from memory (*forget gate*), and one that decides what to output from memory (*output gate*).

k-Nearest Neighbor Learning

k-nearest neighbor learning represents the set of considered functions with all training examples rather than parameters. There is nothing to be learned, but more effort has to be spent when determining the value of the label of an unlabeled example, which involves finding the k training examples most similar to it and then letting them vote on the value.

Support Vector Machine Learning

Linear *support vector machine learning* considers the same functions as single perceptrons in NNs but represents them differently, namely with the subset of the training examples that are closest to the separating plane (*support vectors*), rather than parameters. Support Vector Machine (SVM) learning improves on k-nearest neighbor learning since it only uses the necessary training examples rather than all of them to represent the learned function. It improves on single perceptrons since its separating plane has the largest possible distances to the training examples (*maximum margin separator*), which makes it more robust to sampling noise. The set of considered functions and thus the learned function can also be more complex by embedding the training examples into higher-dimensional spaces with the *kernel trick*.

2.4.2 Unsupervised Learning

In *unsupervised learning*, an agent is provided with unlabeled examples, and its task is to identify structure in the examples, for instance, to find simpler representations of the examples while preserving as much information contained in them as possible (*dimensionality reduction*, which compresses the information in a lossy way and thus might decrease the amount of computation required to process the examples and reduce overfitting), to group similar examples such as photos that contain similar animals (*clustering*), or to generate new examples that are similar to the given ones (*generative modeling*). We describe one unsupervised machine learning technique for each of these purposes, namely principal component analysis for dimensionality reduction, k-means clustering for clustering, and generative adversarial networks for generative modeling. The unlabeled examples can again be partitioned into training, validation, and test examples.

Principal Component Analysis

Principal Component Analysis (PCA) obtains a simpler representation of the training examples by projecting them onto the first principal components to reduce their number of features. The *principal components* are new features that are constructed as linear combinations (or mixtures) of the features such that the new features are uncorrelated and contain most of the information in the training examples (that is, preserve most of the variance). Dimensionality reduction of examples can make them easier to visualize, process, and therefore also analyze, which addresses the curse of dimensionality.

k-Means Clustering

k-means clustering partitions the training examples into k sets of similar examples, see Figure 15. It randomly chooses k different centroids, one for each set, and then repeats the following two steps until the sets no longer change: First, it assigns each

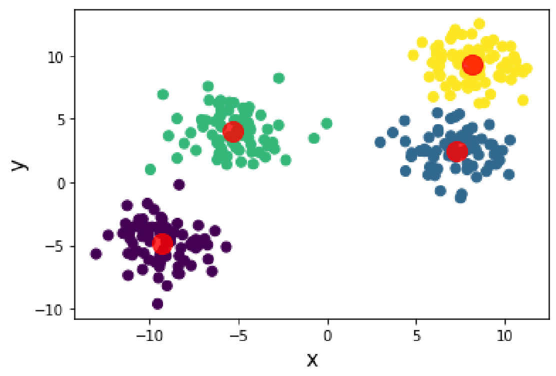


Fig. 15: Clustering with the 4-means clustering technique on training examples (shown as small colored dots) with two real-valued features x and y. Training examples in the same cluster are shown in same color. The centroids of the clusters are shown as large red dots.

example to the set with the least squared Euclidean distance to its centroid. Second, it updates the centroids of the sets to the means of all examples assigned to them.

Generative Adversarial Network Learning

Generative adversarial network learning generates new examples that are similar to the training examples. *Generative Adversarial Networks* (GANs) consist of two NNs that are trained simultaneously: The *generator* attempts to generate examples that are similar to the training examples, and the *discriminator* attempts to distinguish the generated examples from the training examples. Once the discriminator fails in its task, the generator has achieved its task and can be used to generate new examples that are similar to the training examples.

2.4.3 Reinforcement Learning

In *Reinforcement Learning* (RL), similar to operant conditioning in behavioral psychology, an agent interacts with the environment by executing actions and receiving feedback (*reinforcement*) in the form of possibly delayed penalties and rewards, which are represented with real-valued costs. Its task is to identify a behavior (in form of a policy) that minimizes its expected total (discounted) plan-execution cost. RL can be used to figure out which actions worked and which did not when the agent failed to achieve its objective after executing several actions in a row (*credit-assignment problem*), for example, when falling down while learning how to ride a bicycle. Genetic programming can be used for RL, but most RL is based on MDPs and relaxes Assumption 2 from Section 2.3.2 by assuming that the probability distributions over the cost and successor state resulting from an action execution in a state are unknown before the action execution but can be observed afterward. *Model-based RL* uses frequencies to estimate these probability distributions from the observed (state, action, cost, successor state) tuples when the agent interacts with the environment by executing actions. It then determines a policy for the resulting MDP. *Model-free RL* directly estimates a policy. *Q-learning*, the most popular model-free RL technique, is a stochastic dynamic programming technique that learns a q-value $q(s,a)$ for each (state, action) pair, that estimates the smallest possible expected total (discounted) plan-execution cost (until the goal state is reached) if plan execution starts with executing action a in state s. All q-values are initially zero. Whenever the agent executes action a in state s and then observes cost c and successor state s', it updates

$$\begin{aligned} q(s,a) \leftarrow \ & (1-\alpha)q(s,a) + \alpha(c + \gamma \min_{a' \text{ executable in } s'} q(s',a')) && (2) \\ = \ & q(s,a) + \alpha(c + \gamma \min_{a' \text{ executable in } s'} q(s',a') - q(s,a)). && (3) \end{aligned}$$

This formula can easily be understood by realizing that $\min_{a' \text{ executable in } s'} q(s',a')$ estimates the smallest possible expected total (discounted) plan-execution cost if plan execution starts in state s'. Thus, the formula calculates a weighted average of the old q-value $q(s,a)$ and the smallest possible expected total (discounted) plan-execution cost after the agent executes action a in state s and then observes cost c and successor state s', see (2). Equivalently, the formula changes $q(s,a)$ by taking a small step in the direction of the smallest possible expected total (discounted) plan-execution cost after the agent executes action a in state s and then observes cost c and successor state s', see (3). The *learning rate* $\alpha > 0$ is a hyperparameter, typically chosen close to zero, for calculating the weighted average or, equivalently, step size. The discount factor $0 < \gamma \leq 1$ is the hyperparameter from Section 2.3.2, typically chosen close to one, that is used as a mathematical convenience. Deep NNs can be used to approximate the q-values and generalize the experience of the agent to situations that it did not experience during learning (*deep RL*). The policy, to be used after learning, maps each state s to the action a with the smallest q-value $q(s,a)$. These exploiting actions utilize the knowledge of the agent to achieve a small expected total (discounted) plan-execution cost. During learning, the agent also needs to execute exploring actions that allow it to gain new knowledge (for example, help it to visit states that it has visited only a small number of times so far) and improve its policy, as was already discussed in Section 2.1 in the context of local search. The agent could tackle this exploration-exploitation problem, for example, by choosing a random action that can be executed in its current state with a small probability $\epsilon > 0$ and the exploiting action otherwise (*ϵ-greedy exploration*).

2.4.4 Example Applications

Decision tree learning has been used for diagnosing faults and monitoring the conditions of industrial machines [63] and identifying good features for such tasks [129].

Random forest learning has been used for diagnosing faults of industrial machines [150, 21] and classifying remote sensing data [7, 81]. Naïve Bayesian learning has been used for diagnosing faults of industrial machines [155] and diseases [32, 140]. CNN learning has been used for segmenting manufacturing defects in images [147], detecting and classifying faults in semiconductor manufacturing [72], and finding good grasp configurations for novel objects [71]. The Dexterity Network Dataset (Dex-Net) and Grasp Quality CNN (GQ-CNN) are used for robust learning-based grasp planning [86, 83, 84, 82, 85]. RNN learning has been used for classifying objects from their motion trajectories to allow self-driving cars to decide which obstacles to avoid [31] and forecasting time series [50], such as taxi demand [149]. PCA has been used for monitoring and diagnosing faults in industrial processes [23, 111]. SVM learning has been used for classifying remote sensing data [114]. k-nearest neighbor learning has been used for detecting faults in semiconductor manufacturing [77, 159]. k-means clustering has been used for understanding climate and meteorological data, including monitoring pollution, identifying sources, and developing effective control and mitigation strategies [42]. GANs have been used for diagnosing faults [48, 19], detecting credit card fraud [34], and creating novel designs from sketches for rapid prototyping [106]. RL has been used for dispatching orders in the semiconductor industry [128], manipulating objects with industrial robots [62], and following lanes for autonomous driving [65]. Machine vision systems, powered by PCA, SVMs, NNs, and decision trees, have been used for the automated quality inspection of fruit and vegetables [27] as well as machine components [107].

3 Combining AI Techniques

AI research has traditionally focused on improving individual AI techniques (sometimes assuming idealized conditions for their use) and thus on the narrow tasks that they can handle. It is often nontrivial to combine them to create agents that solve broad jobs, for example, perform search-and-rescue operations. For example, it makes sense to combine AI techniques that acquire knowledge (like vision or machine learning) with AI techniques that use it (like planning). However, it is difficult to figure out how different AI techniques should pass information and control back and forth between them in order to achieve synergistic interactions. For example, the combination of low-level motion planning in continuous state spaces and high-level task planning in discrete state spaces has been studied for a long time in the context of using robots to re-arrange objects [61], but no consensus has been reached yet on good ways of combining the two planners. Of course, many complete agents have been built. While one sometimes attempts to determine a monolithic function for such agents directly, one often provides the function as a given composition of subfunctions, resulting in modular agent architectures. Such agent architectures describe how agents are composed of modules, what the modules do, and how they interact. Many such agent architectures are ad hoc, but proposals have been made for general agent architectures, mostly in the context of cognitive agents and robots.

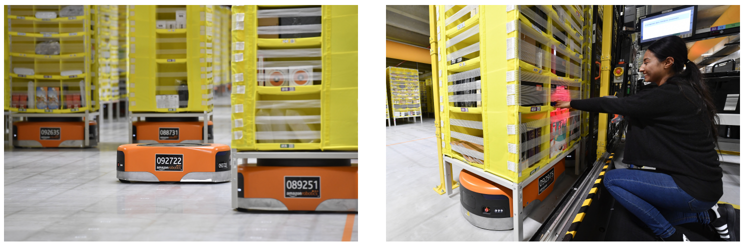


Fig. 16: Amazon fulfillment center. Photos ©Amazon.com, Inc..

For example, the *three-layer robot architecture* consists of a slow planner, a behavior sequencer that always chooses the current behavior during plan execution based on the plan, and a fast reactive feedback controller that implements the chosen behavior. Other agent architectures include the blackboard architecture and the subsumption architecture. Meta-reasoning techniques can be used to decide which output quality a module needs to provide or for how long it should run to allow the agent to maximize its performance. For example, an *anytime algorithm* is one that provides an output quickly and then improves the quality of its output the longer it runs, which can be described by a mapping from its input quality and runtime to its output quality (*performance profile*). It might simplify meta-reasoning if every module of an agent architecture could be implemented as an anytime algorithm.

4 Case Study: Automated Warehousing

In Amazon fulfillment centers, millions of items are stored on special shelves, see Figure 16. When an order needs to be fulfilled, autonomous robots pick up the shelves that store the ordered items and bring them to picking stations at the perimeter of the fulfillment center, where a human worker takes the ordered items from the shelves so that they can be boxed and shipped to the customer. The robots then return the shelves, either to their original locations or different locations [148]. On the order of one thousand robots can operate in such *automated warehouses* and similar *sorting centers* (where robots carry packages to chutes that each serve a different loading dock) per floor [2]. Each robot moves one shelf at a time and recharges when necessary. Amazon puts stickers onto the floor that delineate a four-neighbor grid, on which the robots move essentially without location uncertainty. Human workers no longer need to move around in such automated warehouses, collecting the right items under time pressure – a physically demanding job whose automation makes it possible to store 40 percent more inventory [3] and at least doubles the picking productivity of the human workers (for example, compared to conveyorized systems) [148]. Many of the manipulation tasks in warehouses cannot be automated yet, but the number of human workers required to operate automated warehouses will be small once they are.

Automated warehouses need to store as many items as possible, resulting in the corridors needed to move the shelves being narrow. Robots carrying shelves cannot pass each other in these corridors, and their movements thus need to be planned carefully, which can be done centrally. Shelves close to the perimeter can be fetched faster than shelves in the center. The following questions arise, among others: 1) How should one move the robots to avoid them obstructing each other and allow them to reach their goal locations as quickly as possible to maximize throughput? 2) Where should one place the corridors to maximize throughput? 3) Which picking station should one use for a given order? 4) Which one of several possible shelves should one fetch to obtain an item for a given order? 5) Which robots should one use to fetch a given shelf for a given order? 6) On which shelves should one put items when restocking items? 7) Where should one place a given shelf to maximize throughput, so that shelves that contain frequently ordered items can be fetched fast? 8) When should one start to process a given order given that different orders have different delivery deadlines? 9) How should one estimate the time that it takes a robot to fetch a shelf and a human worker to pick an item, especially since the picking time varies over the course of a shift?

AI techniques, such as those for multi-agent systems, can help make these decisions. For example, the layout optimization problem of Question 2 can be solved with GAs, and the scheduling problems of Questions 3-8 can be solved optimally with MIP techniques or suboptimally with versions of hill climbing. Possible techniques for Question 1 have been studied in the context of the NP-hard *multi-agent path finding* (MAPF) problem, and are described in more detail below.

Centralized MAPF planning techniques plan paths for all robots. Efficient techniques exist but do not result in movement plans with quality guarantees. Examples include techniques based on either movement rules [28, 144] or planning paths for one robot after the other, where each path avoids collisions with the paths already planned (*prioritized search*) [79]. Search can also be used to find movement plans with quality guarantees, but its runtime can then scale exponentially in the number of robots [154, 80]. For example, searching a graph that represents tuples of locations (one for each robot) with vertices is prohibitively slow. Instead, one often divides the overall MAPF problem into mostly independent subproblems by planning a shortest path for each robot under the assumption that the other robots do not exist. If the resulting movement plan has no collisions, then it is optimal. Otherwise, one needs to resolve the collisions. One collision-resolution technique groups all colliding robots into a team and plans for them jointly [126, 127], thus avoiding collisions among them in the future. Another collision-resolution technique chooses one of the collisions, for example, one where two robots are in the same location x at the same time t [121]. It then chooses one of two robots and recursively considers two cases, namely one where the chosen robot is not allowed to be at location x at time t and one where it must be at location x at time t and the other robots are not allowed to be at location x at time t (*disjoint splitting*) [76], thus avoiding this collision in the future. The runtime of the search depends on the choice of collision and robot, but choosing them well is poorly understood and can be slow. Machine learning can be used to make good choices fast [55]. Finally, movement plans with and without quality guarantees can also be found by translating MAPF instances into different optimization instances, such as IPs [153] or SAT instances [130].

Decentralized (and distributed) MAPF planning techniques allow each robot to plan for itself, which avoids the issue of centralized planning that a multi-agent system fails if the central planner fails. One decentralized MAPF planning technique uses deep RL to learn a policy that maps the current state to the next movement of the robot, similar to how Deepmind (which is now part of Google) learned to play Atari video games [93]. The state is characterized by information such as the goal location of the robot, the locations of the other robots and their goal locations, and the locations of the obstacles – all in a field of view centered on the location of the robot. All robots use the same policy. Learning it takes time but needs to be done only once. Afterward, it allows for the quick retrieval of the next movement of the robot based on the current state. A combination of deep RL and learning a policy that imitates the movement plans found by a search technique (*imitation learning*) works even better but is still incomplete [117].

5 AI History

The state-of-the-art in AI, as described so far, has evolved over a period of more than 60 years. In the 1940s and 1950s, researchers worked on creating artificial brains. In 1950, Alan Turing published his paper "Computing Machinery and Intelligence" [136], that asked whether machines could think and introduced the *imitation game* (*Turing test*) to determine whether a computer is intelligent or, in our terminology, a believable agent, namely if a human judge who corresponds via text messages with two conversation partners, a human participant and a computer, cannot determine who is who. In 1956, the term AI was introduced at the Dartmouth conference to name the newly created research field. AI was driven by the belief that "a physical symbol system has the necessary and sufficient means for general intelligent action" [99], that logic provides a good means for knowledge representation, and that reasoning can be achieved with search. In 1958, the AI programming language Lisp was introduced. In the 1960s, AI thrived, but, in the 1970s, the first period of pessimism characterized by reduced funding and interest in AI (*AI winter*) hit due to lack of progress caused by the limited scalability of AI systems and the limited expressiveness of perceptrons. In the 1980s, AI thrived again, due to expert systems and their applications utilizing hand-coded knowledge to overcome the scalability issues, starting an era of knowledge-intensive systems. In the 1980s, the backpropagation technique was re-invented and broadly discovered. It allowed one to train networks of perceptrons, overcoming the limited expressiveness of single perceptrons, starting the research field of *connectionism*, that provided an alternative to the physical symbol system hypothesis. In the late 1980s and early 1990s, the second AI winter hit due to a slowdown in the deployment of expert systems (since it was challenging to build expert systems for complex domains with uncertainty) and the collapse of the Lisp machine market (since many AI systems can easily be implemented with conventional programming languages on conventional computers). In the 1990, AI started to thrive again, due to probabilistic reasoning and planning replacing deterministic logic-based reasoning and planning, and machine learning replacing knowledge acquisition via the work-intensive and error-prone interviewing of experts and hand-coding of their knowledge. Lots of data became available due to the networking of computers via the internet and the pervasiveness of mobile devices, starting an era of *big data* and compute-intensive machine learning techniques to understand and exploit them, such as deep learning with CNNs, resulting in many new AI applications. In fact, it was argued that a substantial increase in the amount of training data can result in much better predictions than improved machine learning techniques [6, 47]. In 2017, the One Hundred Year Study on AI, a long-term effort to study and predict how AI will affect society, issued its first AI Index Report. Its reports show that the number of annual AI publications increased around nine-fold from 1996 to 2017 [122] and the attendance at the machine learning conference NeurIPS increased about eight-fold from 2012 to 2019 [103]. The funding for AI startups increased more than thirty-fold from 2010 to 2018 worldwide [103]. As a result, the percentage of AI-related jobs among all jobs in the US increased about five-fold from 2010 to 2019 [103].

6 AI Achievements

The strength of game-playing AI is often used to evaluate the progress of AI as a whole, probably because humans can demonstrate their intellectual strength by winning games. Many initial AI successes were for board games. In the late 1950s, a Checkers program achieved strong amateur level [115]. In the 1990s, the Othello program BILL [73] defeated the highest-ranked US player, the Checkers program Chinook [118] defeated the world champion, and the chess program DeepBlue [18] defeated the world champion as well [103]. There have been lots of AI successes for non-board games as well. In the 2010s, the Jeopardy! program Watson beat two champions of this question-answer game show [33]. The Poker programs Libratus [15] and DeepStack [95] beat professional players of this card game [103]. A program that played 49 Atari video games demonstrated human-level performance at these early video games by learning directly from video frames [93], the Go program AlphaGo beat one of the world's best Go players of this board game [123], and the Starcraft II program AlphaStar reached Grandmaster level at this modern video game [141].

AI systems now affect all areas of everyday life, industry, and beyond, with especially large visibility in the areas of autonomous driving and intelligent voice assistants. In the 1990s, the Remote Agent controlled a NASA spacecraft without human supervision [11]. Also in the 1990s, a car drove from Pittsburgh to San Diego, with only 2% of the 2,850 miles steered by hand [58]. In the 2000s, several cars autonomously completed the 132-mile off-road course of the second DARPA Grant Challenge and the subsequent DARPA Urban challenge, that required them to drive in realistic urban traffic and perform complex maneuvers [16, 94, 134]. In 2018, Waymo cars autonomously drove over 10 million miles on public roads [70], and, in 2020, Waymo announced that it would open its fully driverless service to the general public [74]. Data from GPS devices and other sensors are now used to estimate current and future traffic conditions [51, 52] and improve public transportation systems [10].

Intelligent voice assistants, such as Amazon Alexa, recognize speech and help their users with a variety of tasks, including finding nearby restaurants and placing orders with them [9, 53].

In 2019, a study found that the diagnostic performance of deep NNs when classifying diseases from medical imaging data had likely matched that of health-care professionals [78].

The future is always difficult to predict. In a survey of all authors of the machine learning conferences ICML and NeurIPS in 2015, Asian respondents predicted that unaided machines would be able to accomplish every individual task better and more cheaply than human workers by 2046 with 50 percent probability, while North American authors predicted that it would happen only by 2090 with 50 percent probability [43].

7 AI Ethics

AI can result in cheaper, better performing, more adaptive, more flexible, and more general automation solutions than more traditional automation techniques. Its increasing commercialization has raised a variety of ethical concerns, as the following examples demonstrate in the context of driverless cars: 1) Adding a small amount of noise to images that is not noticeable by humans can change image recognition results dramatically, which might result in driverless cars not recognizing stop signs that have been slightly altered, for example, unintentionally with dirt or intentionally with chewing gum [100]. 2) Driverless trucks might replace millions of truck drivers in the future [68]. AI also creates jobs, but they often require different skill sets than the eliminated ones [145]. 3) Driverless cars might have to make split-second decisions on how to avoid accidents. If a child suddenly runs onto the road, they might have to decide whether to hit the child or avoid it and potentially injure their passengers (*trolley problem* [5, 132]), which ideally requires explicit ethical judgment on their part [30]. Of course, other applications have also raised concerns: 1) Microsoft's learning chatbot Tay made racist remarks after less than a day on Twitter [90], and 2) a tinder chatbot promoted the movie "Ex Machina" by pretending to be a girl on an online dating site [29]. 3) Significantly fewer women than men were shown online ads promoting well-paying jobs [87], and 4) decision-support systems wrongly labeled more African-American than Caucasian arrested people as potential re-offenders [104], which affects their bail bonds. Overall, AI systems can process large quantities of data, detect regularities in them, draw inferences from them, and determine effective courses of action – sometimes as part of hardware that is able to perform many different, versatile, and potentially dangerous actions. The behavior of AI systems can also be difficult to validate, predict, or explain since they are complex, reason in ways different from humans, and can change their behavior with learning. Finally, their behavior can also be difficult to monitor by humans in case of fast decisions, such as buy and sell decisions on stock markets. Therefore, one needs to worry about the reliability, robustness, and safety of AI systems, provide oversight of their operation, ensure that their behavior is consistent with social norms and human values, determine who is liable for their decisions, and ensure that they impact the standard of living, distribution, and quality of work and other social and economic aspects in a positive way. (The previous four sentences were rephrased from [17].) The World Economic Forum, for example, has identified the following nine important ethical issues raised by AI [13]:1) Unemployment. What happens after the end of jobs? 2) Inequality: How do we distribute the wealth created by machines? 3) Humanity: How do machines affect our behavior and interaction? 4) Artificial stupidity: How can we guard against mistakes? 5) Racist robots: How do we eliminate AI bias? 6) Security: How do we keep AI safe from adversaries? 7) Evil genies: How do we protect against unintended consequences? 8) Singularity: How do we stay in control of a complex intelligent system? 9) Robot rights: How do we define the humane treatment of robots?

These issues have resulted in the AI community focusing more on the explainability and fairness of decisions made by AI systems and starting conferences such as the AAAI/ACM AI, Ethics, and Society (AIES) conference, policy makers trying to regulate AI and its applications, and the IEEE Global Initiative on Ethics of Autonomous and Intelligent Systems creating guidelines and standards for "ethically aligned design" [22] to help designers and developers with the creation of AI systems and safeguard them from liability. In general, philosophy has studied ethics for a long time and the resulting theories apply in this context as well, such as deontology (law-based ethics, as exemplified by Asimov's three laws of robotics [4]), consequentialism (utilitarian ethics), and teleological ethics (virtue ethics) [152].

Further Reading

See [112], the most popular and very comprehensive AI textbook, for in-depth information on AI and its techniques. We followed it in our emphasis of rational agents and the description of the history of AI in Section 5. See [56, 88, 92] for additional AI applications in automation.

Acknowledgements

We thank Zhe Chen, Taoan Huang, Shariq Iqbal, Nitin Kamra, Zhanhao Xiao, and Han Zhang for helpful comments on a draft of this chapter. We also thank Amazon Robotics for providing us with photos of one of their fulfillment centers. Our research was supported by the National Science Foundation (NSF) under grant numbers 1409987, 1724392, 1817189, 1837779, and 1935712 as well as a gift from Amazon. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the sponsoring organizations, agencies, or the U.S. government.

References

1. Intelligence. Merriam-Webster (2020). URL `https://www.merriam-webster.com/dictionary/intelligence`. Accessed 27 October 2020
2. Ackerman, E.: Amazon uses 800 robots to run this warehouse. IEEE Spectrum (2019). Accessed 27 October 2020
3. Amazon Staff: What robots do (and don't do) at Amazon fulfillment centers. The Amazon blog dayone (2020). Accessed 27 October 2020
4. Asimov, I.: Runaround. I, Robot p. 40 (1950)
5. Awad, E., Dsouza, S., Kim, R., Schulz, J., Henrich, J., Shariff, A., Bonnefon, J.F., Rahwan, I.: The moral machine experiment. Nature **563**, 59–64 (2018)
6. Banko, M., Brill, E.: Scaling to very very large corpora for natural language disambiguation. In: Annual Meeting of the Association for Computational Linguistics, pp. 26–33 (2001)
7. Belgiu, M., Drăguţ, L.: Random forest in remote sensing: A review of applications and future directions. ISPRS Journal of Photogrammetry and Remote Sensing **114**, 24–31 (2016)
8. Belov, G., Czauderna, T., de la Banda, M., Klapperstueck, M., Senthooran, I., Smith, M., Wybrow, M., Wallace, M.: Process plant layout optimization: Equipment allocation. In: International Conference on Principles and Practice of Constraint Programming, pp. 473–489 (2018)
9. Berezina, K., Ciftci, O., Cobanoglu, C.: Robots, artificial intelligence, and service automation in restaurants. In: Robots, Artificial Intelligence, and Service Automation in Travel, Tourism and Hospitality, pp. 185–219. Emerald Publishing Limited (2019)
10. Berlingerio, M., Calabrese, F., Di Lorenzo, G., Nair, R., Pinelli, F., Sbodio, M.L.: AllAboard: A system for exploring urban mobility and optimizing public transport using cellphone data. In: Joint European Conference on Machine Learning and Knowledge Discovery in Databases, pp. 663–666 (2013)
11. Bernard, D., Dorais, G., Fry, C., Jr, E., Kanefsky, B., Kurien, J., Millar, W., Muscettola, N., Nayak, P., Pell, B., Rajan, K., Rouquette, N., Smith, B., Williams, B.: Design of the Remote Agent experiment for spacecraft autonomy. In: IEEE Aerospace Conference, pp. 259–281 (1998)
12. Booth, K.E.C., Tran, T.T., Nejat, G., Beck, J.C.: Mixed-integer and constraint programming techniques for mobile robot task planning. IEEE Robotics and Automation Letters **1**(1), 500–507 (2016)
13. Bossmann, J.: Top 9 ethical issues in artificial intelligence. World Economic Forum (2016). Accessed 27 October 2020
14. Boyd, S., Vandenberghe, L.: Convex Optimization, pp. 146–156. Cambridge University Press (2004)
15. Brown, N., Sandholm, T., Machine, S.: Libratus: The superhuman AI for no-limit Poker. In: International Joint Conference on Artificial Intelligence, pp. 5226–5228 (2017)
16. Buehler, M., Iagnemma, K., Singh, S.: The DARPA urban challenge: Autonomous vehicles in city traffic. Springer (2009)
17. Burton, E., Goldsmith, J., Koenig, S., Kuipers, B., Mattei, N., Walsh, T.: Ethical considerations in artificial intelligence courses. AI Magazine **38**(2), 22–34 (2017)
18. Campbell, M., Hoane, A.J., Hsu, F.h.: Deep Blue. Artificial Intelligence **134**(1–2), 57–83 (2002)
19. Cao, S., Wen, L., Li, X., Gao, L.: Application of generative adversarial networks for intelligent fault diagnosis. In: IEEE International Conference on Automation Science and Engineering, pp. 711–715 (2018)
20. Carbery, C.M., Woods, R., Marshall, A.H.: A Bayesian network based learning system for modelling faults in large-scale manufacturing. In: IEEE International Conference on Industrial Technology, pp. 1357–1362 (2018)
21. Cerrada, M., Zurita, G., Cabrera, D., Sánchez, R.V., Artés, M., Li, C.: Fault diagnosis in spur gears based on genetic algorithm and random forest. Mechanical Systems and Signal Processing **70–71**, 87–103 (2016)

22. Chatila, R., Havens, J.C.: The IEEE global initiative on ethics of autonomous and intelligent systems. In: Robotics and Well-Being, pp. 11–16. Springer (2019)
23. Chen, Q., Wynne, R., Goulding, P., Sandoz, D.: The application of principal component analysis and kernel density estimation to enhance process monitoring. Control Engineering Practice **8**(5), 531–543 (2000)
24. Chowanda, A., Blanchfield, P., Flintham, M., Valstar, M.F.: ERiSA: Building emotionally realistic social game-agents companions. In: International Conference on Intelligent Virtual Agents, pp. 134–143 (2014)
25. Croce, F., Delfino, B., Fazzini, P.A., Massucco, S., Morini, A., Silvestro, F., Sivieri, M.: Operation and management of the electric system for industrial plants: An expert system prototype for load-shedding operator assistance. IEEE Transactions on Industry Applications **37**(3), 701–708 (2001)
26. Cropp, M.: Virtual healthcare assistant for the elderly piques interest. Radio New Zealand (2019). Accessed 27 October 2020
27. Cubero, S., Aleixos, N., Moltó, E., Gómez-Sanchis, J., Blasco, J.: Advances in machine vision applications for automatic inspection and quality evaluation of fruits and vegetables. Food and bioprocess technology **4**(4), 487–504 (2011)
28. de Wilde, B., ter Mors, A., Witteveen, C.: Push and rotate: Cooperative multi-agent path planning. In: International Conference on Autonomous Agents and Multi-Agent Systems, pp. 87–94 (2013)
29. Edwards, C., Edwards, A., Spence, P.R., Westerman, D.: Initial interaction expectations with robots: Testing the human-to-human interaction script. Communication Studies **67**(2), 227–238 (2016)
30. Etzioni, A., Etzioni, O.: Incorporating ethics into artificial intelligence. The Journal of Ethics **21**(4), 403–418 (2017)
31. Fathollahi, M., Kasturi, R.: Autonomous driving challenge: To infer the property of a dynamic object based on its motion pattern. In: European Conference on Computer Vision, pp. 40–46 (2016)
32. Fatima, M., Pasha, M.: Survey of machine learning algorithms for disease diagnostic. Journal of Intelligent Learning Systems and Applications **9**(1), 1–16 (2017)
33. Ferrucci, D., Brown, E., Chu-Carroll, J., Fan, J., Gondek, D., Kalyanpur, A.A., Lally, A., Murdock, J.W., Nyberg, E., Prager, J., Schlaefer, N., Welty, C.: Building Watson: An overview of the DeepQA project. AI Magazine **31**(3), 59–79 (2010)
34. Fiore, U., De Santis, A., Perla, F., Zanetti, P., Palmieri, F.: Using generative adversarial networks for improving classification effectiveness in credit card fraud detection. Information Sciences **479**, 448–455 (2019)
35. Fitting, M.: First-Order Logic and Automated Theorem Proving, second edn. Springer (1996)
36. Floudas, C.A., Lin, X.: Mixed integer linear programming in process scheduling: Modeling, algorithms, and applications. Annals of Operations Research **139**(1), 131–162 (2005)
37. Fox, M., Long, D.: PDDL2.1: An extension to PDDL for expressing temporal planning domains. Journal of Artificial Intelligence Research **20**, 61–124 (2003)
38. Gerevini, A., Haslum, P., Long, D., Saetti, A., Dimopoulos, Y.: Deterministic planning in the fifth international planning competition: PDDL3 and experimental evaluation of the planners. Artificial Intelligence **173**(5-6), 619–668 (2009)
39. Ghallab, M., Knoblock, C., Wilkins, D., Barrett, A., Christianson, D., Friedman, M., Kwok, C., Golden, K., Penberthy, S., Smith, D., Sun, Y., Weld, D.: PDDL - the planning domain definition language. Tech. rep., Yale Center for Computational Vision and Control (1998)
40. Gielen, G.G.E., Walscharts, H.C.C., Sansen, W.: Analog circuit design optimization based on symbolic simulation and simulated annealing. IEEE Journal of Solid-State Circuits **25**(3), 707–713 (1990)
41. Godart, F.C., Claes, K.: Semantic networks and the market interface: Lessons from luxury watchmaking. In: Research in the Sociology of Organizations, Structure, Content and Meaning of Organizational Networks, pp. 113–141. Emerald Publishing Limited (2017)

42. Govender, P., Sivakumar, V.: Application of k-means and hierarchical clustering techniques for analysis of air pollution: A review (1980–2019). Atmospheric Pollution Research **11**(1), 40–56 (2020)
43. Grace, K., Salvatier, J., Dafoe, A., Zhang, B., Evans, O.: Viewpoint: When will AI exceed human performance? evidence from AI experts. Journal of Artificial Intelligence Research **62**, 729–754 (2018)
44. Gratch, J., DeVault, D., Lucas, G.M., Marsella, S.: Negotiation as a challenge problem for virtual humans. In: International Conference on Intelligent Virtual Agents, pp. 201–215 (2015)
45. Grebitus, C., Bruhn, M.: Analyzing semantic networks of pork quality by means of concept mapping. Food Quality and Preference **19**(1), 86–96 (2008)
46. Haber, R.E., Alique, J.R., Alique, A., Hernández, J., Uribe-Etxebarria, R.: Embedded fuzzy-control system for machining processes: Results of a case study. Computers in Industry **50**(3), 353–366 (2003)
47. Halevy, A.Y., Norvig, P., Pereira, F.: The unreasonable effectiveness of data. IEEE Intelligent System **24**(2), 8–12 (2009)
48. Han, T., Liu, C., Yang, W., Jiang, D.: A novel adversarial learning framework in deep convolutional neural network for intelligent diagnosis of mechanical faults. Knowledge-Based Systems **165**, 474–487 (2019)
49. Helmert, M., Lasinger, H.: The scanalyzer domain: Greenhouse logistics as a planning problem. In: International Conference on Automated Planning and Scheduling, pp. 234–237 (2010)
50. Hewamalage, H., Bergmeir, C., Bandara, K.: Recurrent neural networks for time series forecasting: Current status and future directions. International Journal of Forecasting **37**(1), 388–427 (2021)
51. Hofleitner, A., Herring, R., Abbeel, P., Bayen, A.M.: Learning the dynamics of arterial traffic from probe data using a dynamic Bayesian network. IEEE Transaction on Intelligent Transportation Systems **13**(4), 1679–1693 (2012)
52. Horvitz, E., Apacible, J., Sarin, R., Liao, L.: Prediction, expectation, and surprise: Methods, designs, and study of a deployed traffic forecasting service. In: Conference on Uncertainty in Artificial Intelligence, pp. 275–283 (2005)
53. Hoy, M.B.: Alexa, Siri, Cortana, and more: An introduction to voice assistants. Medical Reference Services Quarterly **37**(1), 81–88 (2018)
54. Huang, H., Tsai, W.T., Paul, R.: Automated model checking and testing for composite web services. In: IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, pp. 300–307 (2005)
55. Huang, T., Dilkina, B., Koenig, S.: Learning to resolve conflicts for multi-agent path finding with conflict-based search. In: IJCAI Workshop on Multi-Agent Path Finding (2020)
56. Ivanov, S., Webster, C.: Robots, Artificial Intelligence, and Service Automation in Travel, Tourism and Hospitality. Emerald Publishing Limited (2019)
57. Jain, N.K., Jain, V., Deb, K.: Optimization of process parameters of mechanical type advanced machining processes using genetic algorithms. International Journal of Machine Tools and Manufacture **47**(6), 900–919 (2007)
58. Jochem, T., Pomerleau, D.: Life in the fast lane: The evolution of an adaptive vehicle control system. AI Magazine **17**(2), 11–50 (1996)
59. Jones, B., Jenkinson, I., Yang, Z., Wang, J.: The use of Bayesian network modelling for maintenance planning in a manufacturing industry. Reliability Engineering & System Safety **95**(3), 267–277 (2010)
60. Jones, D.S., Fisher, J.N.: Linear programming aids decisions on refinery configurations. In: Handbook of petroleum processing, pp. 1333–1347. Springer (2006)
61. Kaelbling, L.P., Lozano-Pérez, T.: Integrated task and motion planning in belief space. The International Journal of Robotics Research **32**(9-10), 1194–1227 (2013)
62. Kalashnikov, D., Irpan, A., Pastor, P., Ibarz, J., Herzog, A., Jang, E., Quillen, D., Holly, E., Kalakrishnan, M., Vanhoucke, V., Levine, S.: Scalable deep reinforcement learning for

vision-based robotic manipulation. In: Conference on Robot Learning, vol. 87, pp. 651–673 (2018)

63. Karabadji, N.E.I., Khelf, I., Seridi, H., Laouar, L.: Genetic optimization of decision tree choice for fault diagnosis in an industrial ventilator. In: Condition Monitoring of Machinery in Non-Stationary Operations, pp. 277–283. Springer (2012)
64. Kautz, H., Selman, B.: Planning as satisfiability. In: European Conference on Artificial Intelligence, pp. 359–363 (1992)
65. Kendall, A., Hawke, J., Janz, D., Mazur, P., Reda, D., Allen, J., Lam, V., Bewley, A., Shah, A.: Learning to drive in a day. In: IEEE International Conference on Robotics and Automation, pp. 8248–8254 (2019)
66. Kenny, P., Parsons, T.D., Gratch, J., Leuski, A., Rizzo, A.A.: Virtual patients for clinical therapist skills training. In: International Workshop on Intelligent Virtual Agents, pp. 197–210 (2007)
67. Kianpisheh, S., Charkari, N.M.: Dynamic power management for sensor node in WSN using average reward MDP. In: International Conference on Wireless Algorithms, Systems, and Applications, pp. 53–61 (2009)
68. Kitroeff, N.: Robots could replace 1.7 million american truckers in the next decade. Los Angeles Times (2016). Accessed 27 October 2020
69. Knepper, R.A., Layton, T., Romanishin, J., Rus, D.: IkeaBot: An autonomous multi-robot coordinated furniture assembly system. In: IEEE International Conference on Robotics and Automation, pp. 855–862 (2013)
70. Krafcik, J.: Where the next 10 million miles will take us. Waymo Blog (2018). Accessed 27 October 2020
71. Kumra, S., Kanan, C.: Robotic grasp detection using deep convolutional neural networks. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 769–776 (2017)
72. Lee, K.B., Cheon, S., Kim, C.O.: A convolutional neural network for fault classification and diagnosis in semiconductor manufacturing processes. IEEE Transactions on Semiconductor Manufacturing **30**(2), 135–142 (2017)
73. Lee, K.F., Mahajan, S.: BILL: A table-based, knowledge-intensive Othello program. Tech. rep., Carnegie-Mellon University (1986)
74. Lee, T.B.: Waymo finally launches an actual public, driverless taxi service. Ars Technica (2020). Accessed 27 October 2020
75. Lemke, C., Budka, M., Gabrys, B.: Metalearning: A survey of trends and technologies. Artificial Intelligence Review **44**(1), 117–130 (2015)
76. Li, J., Harabor, D., Stuckey, P.J., Ma, H., Koenig, S.: Disjoint splitting for multi-agent path finding with conflict-based search. In: International Conference on Automated Planning and Scheduling, pp. 279–283 (2019)
77. Li, Y., Zhang, X.: Diffusion maps based k-nearest-neighbor rule technique for semiconductor manufacturing process fault detection. Chemometrics and Intelligent Laboratory Systems **136**, 47–57 (2014)
78. Liu, X., Faes, L., Kale, A., Wagner, S., Fu, D., Bruynseels, A., Mahendiran, T., Moraes, G., Shamdas, M., Kern, C., Ledsam, J., Schmid, M., Balaskas, K., Topol, E., Bachmann, L., Keane, P., Denniston, A.: A comparison of deep learning performance against health-care professionals in detecting diseases from medical imaging: A systematic review and meta-analysis. The Lancet Digital Health **1**(6), 271–297 (2019)
79. Ma, H., Harabor, D., Stuckey, P., Li, J., Koenig, S.: Searching with consistent prioritization for multi-agent path finding. In: AAAI Conference on Artificial Intelligence, pp. 7643–7650 (2019)
80. Ma, H., Tovey, C., Sharon, G., Kumar, T.K.S., Koenig, S.: Multi-agent path finding with payload transfers and the package-exchange robot-routing problem. In: AAAI Conference on Artificial Intelligence, pp. 3166–3173 (2016)
81. Ma, L., Cheng, L., Li, M., Liu, Y., Ma, X.: Training set size, scale, and features in geographic object-based image analysis of very high resolution unmanned aerial vehicle imagery. ISPRS Journal of Photogrammetry and Remote Sensing **102**, 14–27 (2015)

82. Mahler, J., Goldberg, K.: Learning deep policies for robot bin picking by simulating robust grasping sequences. In: Annual Conference on Robot Learning, pp. 515–524 (2017)
83. Mahler, J., Liang, J., Niyaz, S., Laskey, M., Doan, R., Liu, X., Ojea, J.A., Goldberg, K.: Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics. In: Robotics: Science and Systems (2017)
84. Mahler, J., Matl, M., Liu, X., Li, A., Gealy, D., Goldberg, K.: Dex-net 3.0: Computing robust robot suction grasp targets in point clouds using a new analytic model and deep learning. In: IEEE International Conference on Robotics and Automation, pp. 1–8 (2018)
85. Mahler, J., Matl, M., Satish, V., Danielczuk, M., DeRose, B., McKinley, S., Goldberg, K.: Learning ambidextrous robot grasping policies. Science Robotics **4**(26), eaau4984 (2019)
86. Mahler, J., Pokorny, F.T., Hou, B., Roderick, M., Laskey, M., Aubry, M., Kohlhoff, K., Kröger, T., Kuffner, J., Goldberg, K.: Dex-net 1.0: A cloud-based network of 3D objects for robust grasp planning using a multi-armed bandit model with correlated rewards. In: IEEE International Conference on Robotics and Automation, pp. 1957–1964 (2016)
87. Martinho-Truswell, E.: As jobs are automated, will men and women be affected equally? Harvard Business Review (2019). Accessed 27 October 2020
88. Masrour, T., Cherrafi, A., Hassani, I.E.: Artificial Intelligence and Industrial Applications: Smart Operation Management. Springer (2021)
89. Matai, R., Singh, S.P., Mittal, M.L.: Traveling salesman problem: An overview of applications, formulations, and solution approaches. Traveling salesman problem, theory and applications **1**, 1–24 (2010)
90. Mathur, V., Stavrakas, Y., Singh, S.: Intelligence analysis of Tay Twitter bot. In: International Conference on Contemporary Computing and Informatics, pp. 231–236 (2016)
91. Mirzaei, A., Moallem, M., Mirzaeian, B., Fahimi, B.: Design of an optimal fuzzy controller for antilock braking systems. In: IEEE Vehicle Power and Propulsion Conference, pp. 823–828 (2005)
92. Mishra, S.K., Polkowski, Z., Borah, S., Dash, R.: AI in Manufacturing and Green Technology, first edn. CRC Press (2021)
93. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529–533 (2015)
94. Montemerlo, M., Thrun, S., Dahlkamp, H., Stavens, D., Strohband, S.: Winning the darpa grand challenge with an ai robot. In: AAAI National Conference on Artificial Intelligence, pp. 982–987 (2006)
95. Moravčík, M., Schmid, M., Burch, N., Lisỳ, V., Morrill, D., Bard, N., Davis, T., Waugh, K., Johanson, M., Bowling, M.: DeepStack: Expert-level artificial intelligence in heads-up no-limit Poker. Science **356**(6337), 508–513 (2017)
96. Munir, A., Gordon-Ross, A.: An MDP-based dynamic optimization methodology for wireless sensor networks. IEEE Transactions on Parallel and Distributed Systems **23**(4), 616–625 (2012)
97. Nägeli, T., Meier, L., Domahidi, A., Alonso-Mora, J., Hilliges, O.: Real-time planning for automated multi-view drone cinematography. ACM Transactions on Graphics **36**(4) (2017)
98. Nannapaneni, S., Mahadevan, S., Rachuri, S.: Performance evaluation of a manufacturing process under uncertainty using Bayesian networks. Journal of Cleaner Production **113**, 947–959 (2016)
99. Newell, A., Simon, H.A.: Computer science as empirical inquiry: Symbols and search. Communication of the ACM **19**(3), 113–126 (1976)
100. Papernot, N., McDaniel, P., Goodfellow, I., Jha, S., Celik, Z.B., Swami, A.: Practical black-box attacks against machine learning. In: ACM on Asia Conference on Computer and Communications Security, pp. 506–519 (2017)
101. Pechoucek, M., Rehak, M., Charvat, P., Vlcek, T., Kolar, M.: Agent-based approach to mass-oriented production planning: Case study. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) **37**(3), 386–395 (2007)

102. Peer, J.: A PDDL based tool for automatic web service composition. In: Principles and Practice of Semantic Web Reasoning, pp. 149–163 (2004)
103. Perrault, R., Shoham, Y., Brynjolfsson, E., Clark, J., Etchemendy, J., Grosz, B., Lyons, T., Manyika, J., Mishra, S., Niebles, J.C.: The AI index 2019 annual report. Tech. rep., Stanford University (2019)
104. Piano, S.L.: Ethical principles in machine learning and artificial intelligence: Cases from the field and possible ways forward. Humanities and Social Sciences Communications **7**(1), 1–7 (2020)
105. Pochet, Y., Wolsey, L.A.: Production planning by mixed integer programming. Springer (2006)
106. Radhakrishnan, S., Bharadwaj, V., Manjunath, V., Srinath, R.: Creative intelligence – automating car design studio with generative adversarial networks (GAN). In: Machine Learning and Knowledge Extraction, pp. 160–175 (2018)
107. Ravikumar, S., Ramachandran, K.I., Sugumaran, V.: Machine learning approach for automated visual inspection of machine components. Expert Systems with Applications **38**(4), 3260–3266 (2011)
108. Reddy, S., Gal, Y., Shieber, S.M.: Recognition of users' activities using constraint satisfaction. In: G.J. Houben, G. McCalla, F. Pianesi, M. Zancanaro (eds.) International Conference on User Modeling, Adaptation, and Personalization, pp. 415–421 (2009)
109. Reyes, A., Ibargüengoytia, P.H., Sucar, L.E.: Power plant operator assistant: An industrial application of factored MDPs. In: Mexican International Conference on Artificial Intelligence, pp. 565–573 (2004)
110. Rintanen, J.: Planning as satisfiability: Heuristics. Artificial Intelligence **193**, 45–86 (2012)
111. Russell, E.L., Chiang, L.H., Braatz, R.D.: Fault detection in industrial processes using canonical variate analysis and dynamic principal component analysis. Chemometrics and Intelligent Laboratory Systems **51**(1), 81–93 (2000)
112. Russell, S.J., Norvig, P.: Artificial intelligence: A modern approach, fourth edn. Pearson (2021)
113. Sakai, Y., Nonaka, Y., Yasuda, K., Nakano, Y.I.: Listener agent for elderly people with dementia. In: ACM/IEEE International Conference on Human-Robot Interaction, pp. 199–200 (2012)
114. Salcedo-Sanz, S., Rojo-Álvarez, J.L., Martínez-Ramón, M., Camps-Valls, G.: Support vector machines in engineering: An overview. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery **4**(3), 234–267 (2014)
115. Samuel, A.L.: Some studies in machine learning using the game of checkers. IBM Journal of Research and Development **3**(3), 211–229 (1959)
116. Santoso, A.F., Supriana, I., Surendro, K.: Designing knowledge of the PPC with semantic network. Journal of Physics: Conference Series **801**, 12015 (2017)
117. Sartoretti, G., Kerr, J., Shi, Y., Wagner, G., Kumar, S., Koenig, S., Choset, H.: PRIMAL: Pathfinding via reinforcement and imitation multi-agent learning. IEEE Robotics and Automation Letters **4**(3), 2378–2385 (2019)
118. Schaeffer, J., Lake, R., Lu, P., Bryant, M.: CHINOOK: The world man-machine Checkers champion. AI Magazine **17**(1), 21–29 (1996)
119. Schmitt, P.H.: First-order logic. In: Deductive Software Verification, pp. 23–47. Springer (2016)
120. Seto, E., Leonard, K.J., Cafazzo, J.A., Barnsley, J., Masino, C., Ross, H.J.: Developing healthcare rule-based expert systems: Case study of a heart failure telemonitoring system. International Journal of Medical Informatics **81**(8), 556–565 (2012)
121. Sharon, G., Stern, R., Felner, A., Sturtevant, N.: Conflict-based search for optimal multi-agent pathfinding. Artificial Intelligence **219**, 40–66 (2015)
122. Shoham, Y., Perrault, R., Brynjolfsson, E., Clark, J., LeGassick, C.: The AI index 2017 annual report. Tech. rep., Stanford University (2017)
123. Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham,

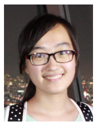
J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., Hassabis, D.: Mastering the game of Go with deep neural networks and tree search. Nature **529**, 484–489 (2016)
124. Soh, L.K., Tsatsoulis, C., Gineris, D., Bertoia, C.: ARKTOS: An intelligent system for SAR sea ice image classification. IEEE Transactions on Geoscience and Remote Sensing **42**(1), 229–248 (2004)
125. Srinivasu, D., Babu, N.R.: A neuro-genetic approach for selection of process parameters in abrasive waterjet cutting considering variation in diameter of focusing nozzle. Applied Soft Computing **8**(1), 809–819 (2008)
126. Standley, T.: Finding optimal solutions to cooperative pathfinding problems. In: AAAI Conference on Artificial Intelligence, pp. 173–178 (2010)
127. Standley, T., Korf, R.: Complete algorithms for cooperative pathfinding problems. In: International Joint Conference on Artificial Intelligence, pp. 668–673 (2011)
128. Stricker, N., Kuhnle, A., Sturm, R., Friess, S.: Reinforcement learning for adaptive order dispatching in the semiconductor industry. CIRP Annals **67**(1), 511–514 (2018)
129. Sugumaran, V., Muralidharan, V., Ramachandran, K.: Feature selection using decision tree and classification through proximal support vector machine for fault diagnostics of roller bearing. Mechanical Systems and Signal Processing **21**(2), 930–942 (2007)
130. Surynek, P.: Reduced time-expansion graphs and goal decomposition for solving cooperative path finding sub-optimally. In: International Joint Conference on Artificial Intelligence, pp. 1916–1922 (2015)
131. Tchertchian, N., Yvars, P.A., Millet, D.: Benefits and limits of a constraint satisfaction problem/life cycle assessment approach for the ecodesign of complex systems: A case applied to a hybrid passenger ferry. Journal of Cleaner Production **42**, 1–18 (2013)
132. Thomson, J.J.: The trolley problem. The Yale Law Journal **94**(6), 1395–1415 (1985)
133. Thrun, S., Burgard, W., Fox, D.: Probabilistic Robotics, chap. Application to Robot Control, pp. 503–507. MIT Press (2005)
134. Thrun, S., Montemerlo, M., Dahlkamp, H., Stavens, D., Aron, A., Diebel, J., Fong, P., Gale, J., Halpenny, M., Hoffmann, G., Lau, K., Oakley, C., Palatucci, M., Pratt, V., Stang, P., Strohband, S., Dupont, C., Jendrossek, L.E., Koelen, C., Markey, C., Rummel, C., van Niekerk, J., Jensen, E., Alessandrini, P., Bradski, G., Davies, B., Ettinger, S., Kaehler, A., Nefian, A., Mahoney, P.: Stanley: The robot that won the DARPA grand challenge. Journal of Field Robotics **23**(9), 661–692 (2006)
135. Tsiourti, C., Moussa, M.B., Quintas, J., Loke, B., Jochem, I., Lopes, J.A., Konstantas, D.: A virtual assistive companion for older adults: Design implications for a real-world application. In: SAI Intelligent Systems Conference, pp. 1014–1033 (2016)
136. Turing, A.M.: Computing Machinery and Intelligence. Mind **LIX**(236), 433–460 (1950)
137. Umarov, I., Mozgovoy, M.: Believable and effective AI agents in virtual worlds: Current state and future perspectives. International Journal of Gaming and Computer-Mediated Simulations **4**(2), 37–59 (2012)
138. Vardoulakis, L.P., Ring, L., Barry, B., Sidner, C.L., Bickmore, T.W.: Designing relational agents as long term social companions for older adults. In: International Conference on Intelligent Virtual Agents, pp. 289–302 (2012)
139. Vecchi, M.P., Kirkpatrick, S.: Global wiring by simulated annealing. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **2**(4), 215–222 (1983)
140. Vembandasamy, K., Sasipriya, R., Deepa, E.: Heart diseases detection using naïve Bayes algorithm. International Journal of Innovative Science, Engineering & Technology **2**(9), 441–444 (2015)
141. Vinyals, O., Babuschkin, I., Czarnecki, W.M., Mathieu, M., Dudzik, A., Chung, J., Choi, D.H., Powell, R., Ewalds, T., Georgiev, P., Oh, J., Horgan, D., Kroiss, M., Danihelka, I., Huang, A., Sifre, L., Cai, T., Agapiou, J.P., Jaderberg, M., Vezhnevets, A.S., Leblond, R., Pohlen, T., Dalibard, V., Budden, D., Sulsky, Y., Molloy, J., Paine, T.L., Gulcehre, C., Wang, Z., Pfaff, T., Wu, Y., Ring, R., Yogatama, D., Wünsch, D., McKinney, K., Smith, O., Schaul, T., Lillicrap, T., Kavukcuoglu, K., Hassabis, D., Apps, C., Silver, D.: Grandmaster level in StarCraft II using multi-agent reinforcement learning. Nature **575**, 350–354 (2019)

142. Wang, C., Wang, L., Qin, J., Wu, Z., Duan, L., Li, Z., Cao, M., Ou, X., Su, X., Li, W., Lu, Z., Li, M., Wang, Y., Long, J., Huang, M., Li, Y., Wang, Q.: Path planning of automated guided vehicles based on improved a-star algorithm. In: IEEE International Conference on Information and Automation, pp. 2071–2076 (2015)
143. Wang, G., Hasani, R.M., Zhu, Y., Grosu, R.: A novel Bayesian network-based fault prognostic method for semiconductor manufacturing process. In: IEEE International Conference on Industrial Technology, pp. 1450–1454 (2017)
144. Wang, K., Botea, A.: MAPP: a scalable multi-agent path planning algorithm with tractability and completeness guarantees. Journal of Artificial Intelligence Research **42**, 55–90 (2011)
145. Wilson, H.J., Daugherty, P.R., Morini-Bianzino, N.: The jobs that artificial intelligence will create. MIT Sloan Management Review **58**(4), 14–16 (2017)
146. Wolpert, D.H., Macready, W.G.: No free lunch theorems for optimization. IEEE Transactions on Evolutionary Computation **1**(1), 67–82 (1997)
147. Wong, V.W., Ferguson, M., Law, K., Lee, Y.T.T., Witherell, P.: Automatic volumetric segmentation of additive manufacturing defects with 3D U-Net. In: AAAI Spring Symposium on AI in Manufacturing (2020)
148. Wurman, P., D'Andrea, R., Mountz, M.: Coordinating hundreds of cooperative, autonomous vehicles in warehouses. AI Magazine **29**(1), 9–20 (2008)
149. Xu, J., Rahmatizadeh, R., Bölöni, L., Turgut, D.: Real-time prediction of taxi demand using recurrent neural networks. IEEE Transactions on Intelligent Transportation Systems **19**(8), 2572–2581 (2018)
150. Yang, B.S., Di, X., Han, T.: Random forests classifier for machine fault diagnosis. Journal of Mechanical Science and Technology **22**(9), 1716–1725 (2008)
151. Younes, H.L.S., Littman, M.L.: PPDDL1.0: an extension to PDDL for expressing planning domains with probabilistic effects. Tech. rep., Carnegie Mellon University (2004)
152. Yu, H., Shen, Z., Miao, C., Leung, C., Lesser, V.R., Yang, Q.: Building ethics into artificial intelligence. In: International Joint Conference on Artificial Intelligence, pp. 5527–5533 (2018)
153. Yu, J., LaValle, S.: Planning optimal paths for multiple robots on graphs. In: IEEE International Conference on Robotics and Automation, pp. 3612–3617 (2013)
154. Yu, J., LaValle, S.: Structure and intractability of optimal multi-robot path planning on graphs. In: AAAI Conference on Artificial Intelligence, pp. 1444–1449 (2013)
155. Yusuf, S., Brown, D.J., Mackinnon, A., Papanicolaou, R.: Fault classification improvement in industrial condition monitoring via hidden Markov models and naïve Bayesian modeling. In: IEEE Symposium on Industrial Electronics & Applications, pp. 75–80 (2013)
156. Yvars, P.A.: Using constraint satisfaction for designing mechanical systems. International Journal on Interactive Design and Manufacturing **2**, 161–167 (2008)
157. Zhao, Y., Collins, E.G.: Fuzzy PI control design for an industrial weigh belt feeder. IEEE Transactions on Fuzzy Systems **11**(3), 311–319 (2003)
158. Zhao, Z., Yu, Z., Sun, Z.: Research on fuzzy road surface identification and logic control for anti-lock braking system. In: IEEE International Conference on Vehicular Electronics and Safety, pp. 380–387 (2006)
159. Zhou, Z., Wen, C., Yang, C.: Fault detection using random projections and k-nearest neighbor rule for semiconductor manufacturing processes. IEEE Transactions on Semiconductor Manufacturing **28**(1), 70–79 (2014)
160. Zolpakar, N.A., Lodhi, S.S., Pathak, S., Sharma, M.A.: Application of multi-objective genetic algorithm (MOGA) optimization in machining processes. In: Optimization of Manufacturing Processes, pp. 185–199. Springer (2020)

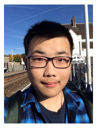
Author Biographies

Sven Koenig is a professor in computer science at the University of Southern California. His research focuses on AI techniques for making decisions that enable single situated agents (such as robots or decision-support systems) and teams of agents to act intelligently in their environments and exhibit goal-directed behavior in real-time, even if they have only incomplete knowledge of their environments, imperfect abilities to manipulate them, limited or noisy perception, or insufficient reasoning speed.

Shao-Hung Chan is a Ph.D. student in computer science at the University of Southern California. He received a Bachelor of Science degree from National Cheng-Kung University in 2017 and a Master of Science degree from National Taiwan University in 2019. His research focuses on designing planning techniques for real-world applications, including hierarchical planning techniques that allow teams of agents to navigate without collisions.



Jiaoyang Li is a Ph.D. student in computer science at the University of Southern California. She received a Bachelor's degree in automation from Tsinghua University in 2017. Her research is on multi-agent systems and focuses on developing efficient planning techniques that enable hundreds of autonomous robots to fulfill navigation requests without collisions.



Yi Zheng is a Ph.D. student in computer science at the University of Southern California. He received a Bachelor of Science degree in computer science from the University of Southampton in 2019. His research focuses on developing scalable planning techniques for multi-agent systems. He is also interested in applying machine learning to multi-agent planning problems.