A COMPUTER IMPLEMENTED METHOD AND SYSTEM FOR AUTOMATICALLY MODELLING A PROBLEM AND ORCHESTRATING CANDIDATE ALGORITHMS TO SOLVE THE PROBLEM

FIELD OF THE INVENTION

[0001] The invention pertains to the field of modelling a problem and using algorithms to solve the problem.

BACKGROUND

[0002] Currently, the common way researchers and data scientists model a “problem” is via a manual, iterative trial and error or ad-hoc process. A “problem” is any kind of problem-solving that requires mathematical modelling to perform calculations and predictions, and selection and deployment of algorithms. In the current method, parameters of the model would be manually initialized with values either from prior knowledge or in a random manner. The resultant output is then studied and if the output is not what is expected, the parameters are manually tweaked and the model resolved. However this is a time-consuming process. Furthermore, the problems to be modelled are usually unique in nature which makes the current manual method even more deficient and ineffective.

[0003] Therefore, there exists a need to develop a system and method that is capable of automatically modelling a problem, and automatically orchestrating candidate algorithms to solve the problem.

SUMMARY OF INVENTION

[0004] According to a first aspect of the invention, a computer implemented method for automatically modelling a problem and orchestrating algorithms to obtain a solution to the problem is described, the method comprising the steps of establishing with an artificial intelligence orchestrator manager module, that a task has to be performed to solve the problem; deriving with a task manager module, at least one subtask from the task; and selecting with an algorithm orchestrator module, a plurality of candidate algorithms based on the at least one subtask. The method further comprises the steps of training with the algorithm orchestrator

module, the plurality of selected candidate algorithms to obtain training statistics; and building with the algorithm orchestrator module, a template POMDP (partially observable markov decision process) model based on the plurality of selected candidate algorithms and the training statistics. The method further comprises the steps of parametrizing with the algorithm orchestrator module, the template POMDP model into POMDP model parameters; and implementing with the algorithm orchestrator module, a genetic algorithm to optimize the POMDP model parameters, wherein the solution to the problem is derived from the optimized POMDP model parameters.

[0005] Preferably, the method further comprises the step of creating with the algorithm orchestrator module, at least one design-of-experiment linear combination of the plurality of selected candidate algorithms; and wherein building the template POMDP model will be based on the plurality of selected candidate algorithms, the training statistics and the at least one design-of-experiment linear combination of the plurality of selected candidate algorithms.

[0006] Preferably, the template POMDP model comprises states, transition probabilities, reward functions and observation functions; and wherein the plurality of selected candidate algorithms and the at least one design-of-experiment linear combination of the plurality of selected candidate algorithms form the states of the template POMDP model, and the transition probabilities, the reward functions, and the observation functions of the template POMDP model are derived from the training statistics.

[0007] Preferably, the step of implementing with the algorithm orchestrator module, a genetic algorithm to optimize the POMDP model parameters comprises the steps of (a) mutating the POMDP model parameters with the genetic algorithm to form sets of mutated POMDP model parameters; (b) instantiating a candidate POMDP model for each set of mutated POMDP model parameters and solve the POMDP candidate model to obtain a policy; (c) determining the value of the policy for each set of mutated POMDP model parameters; (d) using a sorting algorithm to rank the sets of mutated POMDP model parameters based on the value of the policy, to obtain a highest ranked set of mutated POMDP model parameters; (e) repeating steps (a), (b), (c) and (d) if a termination criterion is not met; and (f) returning the highest ranked set of mutated POMDP model parameters as the optimized POMDP model parameters if the termination criterion is met.

[0008] Preferably, the termination criterion is when a fixed number of iterations has ran or when a threshold value for the policy has been reached.

[0009] Preferably, the sorting algorithm is non-dominated sorting.

[0010] Preferably, the method further comprises the step of receiving with a universal model module, the domain knowledge and types of problems that can be solved by a user-application.

[0011] Preferably, the method further comprises the step of receiving with the artificial intelligence orchestrator manager module, user data from the user-application, the user data specifying the problem to be solved.

[0012] Preferably, the method further comprises the steps of checking and ensuring the data integrity of the user data with a data manager module, and providing the data integral user data to algorithm orchestrator module for training the plurality of selected candidate algorithms.

[0013] Preferably, the step of selecting with an algorithm orchestrator module, a plurality of candidate algorithms based on the at least one subtask comprises the steps of consulting an algorithm foundation knowledge module and selecting the plurality of candidate algorithms that correspond to the at least one subtask; and instantiating concrete instances of the plurality of selected candidate algorithms from an algorithm pool module.

[0014] Preferably, the method further comprises the step of storing the training statistics in an algorithm self-learning knowledge module.

[0015] Preferably, the plurality of selected candidate algorithms are any combination of the following: POMDP, genetic algorithm, artificial neural networks, natural language processing, fuzzy logic, Apiori, k-means clustering, adaptive boosting, support vector machine, naive Bayes and logistics regression.

[0016] According to a second aspect of the invention, a computer program product embodied on a computer readable medium containing instructions to perform the computer implemented method is described.

[0017] According to a third aspect of the invention, a system for automatically modelling a problem and orchestrating algorithms to obtain a solution to the problem is described, the system comprising at least one processor programmed to implement an artificial intelligence orchestrator manager module, a task manager module and an algorithm orchestrator module. Wherein the artificial intelligence orchestrator manager module is configured to establish that a task has to be performed to solve the problem; and wherein the task manager module is configured to derive at least one subtask from the task. Wherein the algorithm orchestrator module is configured to select a plurality of candidate algorithms based on the at least one subtask; train the plurality of selected candidate algorithms to obtain training statistics; build a template POMDP (partially observable markov decision process) model based on the plurality of selected candidate algorithms and the training statistics; parametrize the template POMDP model into POMDP model parameters; and implement a genetic algorithm to optimize the POMDP model parameters, wherein the solution to the problem is derived from the optimized POMDP model parameters.

[0018] Preferably, the algorithm orchestrator module is further configured to create at least one design-of-experiment linear combination of the plurality of selected candidate algorithms; and build the template POMDP model based on the plurality of selected candidate algorithms, the training statistics and the at least one design-of-experiment linear combination of the plurality of selected candidate algorithms.

[0019] Preferably, the template POMDP model comprises states, transition probabilities, reward functions and observation functions; and wherein the plurality of selected candidate algorithms and the at least one design-of-experiment linear combination of the plurality of selected candidate algorithms form the states of the template POMDP model, and the transition probabilities, the reward functions, and the observation functions of the template POMDP model are derived from the training statistics.

[0020] Preferably, the algorithm orchestrator module is further configured to (a) mutate the POMDP model parameters with the genetic algorithm to form sets of mutated POMDP model parameters; (b) instantiate a candidate POMDP model for each set of mutated POMDP model parameters and solve the POMDP candidate model to obtain a policy; (c) determine the value of the policy for each set of mutated POMDP model parameters; (d) use a sorting algorithm to rank the sets of mutated POMDP model parameters based on the value of the policy, to obtain a highest ranked set of mutated POMDP model parameters; (e) repeat

steps (a), (b), (c) and (d) if a termination criterion is not met; and (f) return the highest ranked set of mutated POMDP model parameters as the optimized POMDP model parameters if the termination criterion is met.

[0021] Preferably, the termination criterion is when a fixed number of iterations has ran or when a threshold value for the policy has been reached.

[0022] Preferably, the sorting algorithm is non-dominated sorting.

[0023] Preferably, the at least one processor is further programmed to implement a universal model module, wherein the universal model module is configured to receive domain knowledge and types of problems that can be solved by a user-application.

[0024] Preferably, the artificial intelligence orchestrator manager module is further configured to receive user data from the user-application, the user data specifying the problem to be solved.

[0025] Preferably, the at least one processor is further programmed to implement a data manager module to check and ensure the data integrity of the user data, and provide the data integral user data to algorithm orchestrator module for training the plurality of selected candidate algorithms.

[0026] Preferably, the at least one processor is further programmed to implement an algorithm foundation knowledge module and an algorithm pool module, and the algorithm orchestrator module selects the plurality of candidate algorithms based on the at least one subtask by consulting the algorithm foundation knowledge module and selecting the plurality of candidate algorithms that correspond to the at least one subtask; and instantiating concrete instances of the plurality of selected candidate algorithms from the algorithm pool module.

[0027] Preferably, the at least one processor is further programmed to implement an algorithm self-learning knowledge module to store the training statistics.

[0028] Preferably, the plurality of selected candidate algorithms are any combination of the following: POMDP, genetic algorithm, artificial neural networks, natural language processing, fuzzy logic, Apiori, k-means clustering, adaptive boosting, support vector machine, naive Bayes and logistics regression.

[0029] The invention will now be described in detail with reference to the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

[0030] The accompanying figures illustrate disclosed embodiment(s) and serve to explain principles of the disclosed embodiment(s). It is to be understood, however, that these drawings are presented for purposes of illustration only, and not for defining limits of the application.

[0031] Figure 1 shows a schematic diagram of a system for automatically modelling a problem and orchestrating candidate algorithms to solve the problem, according to an exemplary embodiment.

[0032] Figure 2 shows a flow chart for automatically modelling a problem specified by an end-user using a user-application and orchestrating candidate algorithms to solve the problem.

[0033] Figure 3 shows a dynamic Bayesian network representation of a Partially Observable Markov Decision Process (POMDP) model.

[0034] Figure 4 shows a dynamic Bayesian network representation of a Partially Observable Markov Decision Process (POMDP) model.

[0035] Figure 5 shows a flow chart depicting the steps taken by algorithm orchestrator module in using a genetic algorithm to optimize the POMDP model parameters.

[0036] Exemplary, non-limiting embodiments of the present application will now be described with references to the above-mentioned figures.

DETAILED DESCRIPTION

[0037] The invention as described herein is a computer implemented method and system that is capable of automatically modelling a problem, and orchestrating candidate

algorithms to solve the problem. Such a system is shown in figure 1. System 100 comprises universal model module 101, artificial intelligence orchestrator manager module 102, task manager module 103, data manager module 104, algorithm orchestrator module 105, algorithm pool module 106, algorithm foundation knowledge module 107 and algorithm self-learning knowledge module 108.

[0038] System 100 can act as an artificial intelligence engine for user-applications. An application developer would have to first enter into system 100 the types of problems the application developer's user-application can solve, as well as the domain knowledge of the type of problems and the expected output. An end-user using the user-application would then enter the user data or (specifications of the problem to be solved) into the user-application, which would then be fed to system 100. System 100 will then automatically model the problem, select candidate algorithms and orchestrate the candidate algorithms to solve the problem. System 100 therefore can function as the "brains" of user-applications in solving problems.

[0039] Universal model module 101 acts as a front-end user interface that allows an application developer to specify the type of problems the application developer's user-application can solve. The application developer can also enter into universal model module 101 the domain knowledge of the type of problems and the expected output. These are the pre-configuration steps that need to be done before system 100 can automate the modelling of the problem. Universal model module 101 is responsible for checking the integrity of the task variables and parameters and flags out to the application developer if these are missing or wrongly labeled. Universal model module 101 can parse the different components of the domain knowledge into machine readable format. This machine readable format can be read and understood by artificial intelligence orchestrator manager module 102 for further processing and computation.

[0040] Artificial intelligence orchestrator manager module 102 acts as the interface between the user-application and system 100. When end-user enters the user data (i.e. specifications of the problem to be solved) into the user-application, the user-application will provide the user data to artificial intelligence orchestrator manager module 102. Artificial intelligence orchestrator manager module 102 then retrieves the domain knowledge of the particular type of problem from universal model module 101. From this, artificial intelligence orchestrator manager module 102 will discern the task that needs to be performed in order to

solve the problem. Artificial intelligence orchestrator manager module 102 will then send this task to task manager module 103. Artificial intelligence orchestrator manager module 102 will also handle resource allocation and decide (depending on availability of CPU resources, urgency of task, parallelizability of the task etc) if a new task can be executed immediately or put on a queue. Artificial intelligence orchestrator manager module 102 can also discern the number of instance licenses that the user-application is entitled to. If the user-application has multiple instance licenses, artificial intelligence orchestrator manager module 102 will solve multiple problems defined by user-application by executing multiple tasks concurrently. Artificial intelligence orchestrator manager module 102 will also manage intermediate/diagnostics reports, for example, keeping track of events or persisting serialized data model. Artificial intelligence orchestrator manager module 102 will also generate a universally unique identifier (UUID) for a particular task. Artificial intelligence orchestrator manager module 102 will also report the estimated running time to complete a task to the end-user.

[0041] The role of task manager module 103 is to coordinate the completion of a given task by initiating subtasks in an ordered sequence. Therefore, when task manager module 103 receives the task from artificial intelligence orchestrator manager module 102, task manager module 103 will derive which subtask(s) needs to be executed. Task manager module 103 will pass the user data to data manager module 104 to perform data integrity checks, and then send the subtask(s) to algorithm orchestrator module 105 for execution. In-between subtasks, task manager module 103 will instruct data manager module 104 on how to process (or merge) the output for the previous subtask for the subsequent subtask. When there are interrupts or exceptions, task manager module 103 can also swap higher priority tasks for lower priority ones. Task manager module 103 can also collate intermediate/diagnostics reports to pass on to artificial intelligence orchestrator manager module 102. Task manager module 103 can also notify artificial intelligence orchestrator manager module 102 when there is missing data or when the subtasks have been completed. Task manager module 103 can also store the serialized current model state, to allow for in-between restarts for each task. Examples of serialized models are shown below.

Example 1: a serialized model for a research problem:

{UUID: a171-2470b63dff00, Task: “businessAnalytics-operationalPerformance”, Subtasks: {“DWM: obtain relevant URLs”, “NLP: relevant text extraction”, “NLP: summarize results”}, DataSource: NULL, Locations: NULL, Type: {“discrete”, “discrete”, “discrete”}, Merging: {“DWM: obtain relevant URLs”: “use results as

input", "NLP: relevant text extraction": "use results as input"}, Restart: TRUE, RestartInfo: {"Continue": "NLP: relevant text extraction", ProcessedDataSource{url: "jdbc:mysql://host:3306/path-to-processed-database", user: "root", password: "mypassword", tableId: "654321"}}, ExternalSource: TRUE, AdditionalFile: NULL};

Example 2: a serialized model for a business analytics problem:

{UUID: d2f9-9af509fb5889, Task: "research", Subtasks: {"clustering", "regression", "prediction"}, DataSource: { SalesDB{url: "jdbc:mysql//host:3306/path-to-database", user: "userName", password: "password123"}, PayrollDB{url: "jdbc:postgresql://host:4434/path-to-database", user: "root", password: "password321"}, Locations: {"clustering": {var1: "SalesDB", "table1", "col2"; var2: "SalesDB", "table2", "col3"; class: "SalesDB", "table4", "col5"}, "regression": {var1: "SalesDB", "table1", "col5"; var2: "PayrollDB", "table2", "col8"; class: "SalesDB", "tabl24", "col18"}, "prediction": {var1: "SalesDB", "table10", "col20"; var2: "SalesDB", "table20", "col30"; class: "SalesDB", "table4", "col5"}}, Type: {"discrete", "discrete", "discrete"}, Merging: {"clustering": "use results as input", "regression": "use results as input"}, Restart: FALSE, RestartInfo: NULL, ExternalSource: TRUE, AdditionalFile: NULL}.

[0042] The role of data manager module 104 is to ensure data integrity. Data manager module 104 can check the user data for missing data or incomplete data. In situations where the user data is inadequate (i.e. end-user failed to provide all the specifications of the problem or has not described the problem completely or had failed to provide or upload supporting/required data) and therefore there is "missing data", data manager module 104 can alert task manager module 103 that data is missing. Data manager module 104 can locate this missing data by "crawling" the web to locate and determine this missing data. In situations where the user data is incomplete (for example, when not all the entries for a particular column in a database are available), data manager module 104 can impute this incomplete data, for example, by using known heuristics. After data integrity has been ensured, data manager module 104 will pass the processed data (after accounting for any missing or incomplete data) to algorithm orchestrator module 105. Data manager module 104 can also collate data from different sources into one common source location for algorithm orchestrator module 105 to access. The collated data can be in a format in which algorithm orchestrator module 105 can process, for example, in ARFF (attribute-relation file format) or in XML (extensible markup language). Data manager module 104 can also store the subtasks' results when directed by task manager module 103.

[0043] Algorithm orchestrator module 105 orchestrates the relevant algorithms required to achieve the particular subtask's objective. Therefore, once algorithm orchestrator module 105 receives the subtask from task manager module 103, algorithm orchestrator module 105 will consult algorithm foundation knowledge module 107 for the candidate algorithms for solving the subtask. Algorithm orchestrator module 105 will then instantiate concrete instances of these selected candidate algorithms from algorithm pool module 106. Algorithm orchestrator module 105 then trains each of the selected candidate algorithms using the processed data received from data manager module 104 to obtain training statistics. Examples of such training statistics non-exhaustively include recall and precision accuracy, n-fold cross validation error, confusion matrix and the like. If historical training statistics exists in algorithm self-learning knowledge module 108, algorithm orchestrator module 105 will also obtain these training statistics from algorithm self-learning knowledge module 108. At the end of the training, algorithm orchestrator module 105 will store the recently acquired training statistics into algorithm self-learning knowledge module 108 such that the training statistics in algorithm self-learning knowledge module 108 accumulate over time.

[0044] Algorithm orchestrator module 105 can also create a design-of-experiment (DOE) range on the different linear combinations of the selected candidate algorithms. Algorithm orchestrator module 105 can then use the selected candidate algorithms, the design-of-experiment (DOE) range on the different linear combinations of the selected candidate algorithms and the training statistics to build a template Partially Observable Markov Decision Process (POMDP) model. A POMDP is a generalization of a Markov decision process (MDP). A POMDP models an agent decision process in which it is assumed that the system dynamics are determined by an MDP, but the agent cannot directly observe the underlying state. Instead, it must maintain a probability distribution over the set of possible states, based on a set of observations and observation probabilities, and the underlying MDP. The selected candidate algorithms and the design-of-experiment (DOE) range on the different linear combinations of the selected candidate algorithms will form the states of the template POMDP model, while the transition probabilities, reward functions and observation probabilities of the template POMDP model are derived from the training statistics.

[0045] Algorithm orchestrator module 105 then parametrizes the template POMDP model into POMDP model parameters (which are essentially numerals). Algorithm orchestrator module 105 then uses a genetic algorithm to optimize the POMDP model

parameters. Algorithm orchestrator module 105 then instantiates a final POMDP model and solves this final POMDP model. The outcome of solving the final POMDP model is a policy. The policy is then run and a "sequence" and "weight" of executing the selected candidate algorithms is obtained. "Sequence" being the order in which the selected candidate algorithms are executed while "weight" being the amount in which the selected candidate algorithms are executed. After execution of the selected candidate algorithms in accordance with the "sequence" and "weight", the output will be the solution to the problem (or in a situation where the task has been broken into multiple subtasks, the output will be a partial solution to the problem).

[0046] Algorithm pool module 106 stores the code of the algorithms available to be orchestrated and deployed. Therefore, when algorithm orchestrator module 105 runs the algorithms, algorithm orchestrator module 105 actually instantiates concrete instances of the algorithms via algorithm pool module 106. These algorithms non-exhaustively include: partially observable markov decision process (POMDP); genetic algorithms; artificial neural networks; natural language processing; fuzzy logic; apiori, k-means clustering, adaptive boosting, support vector machine, naive Bayes, logistics regression and the like.

[0047] Algorithm foundation knowledge module 107 stores the domain knowledge of each algorithm available in algorithm pool module 106. In particular, the domain knowledge comprises the candidate algorithms suitable for solving a subtask. For example, algorithm foundation knowledge module 107 may have the information that for the subtask "clustering", the Apriori algorithm and k-means clustering are suitable algorithms for performing this subtask (i.e. candidate algorithms). In another example, algorithm foundation knowledge module 107 may have the information that for the subtask "classification", the adaptive boosting algorithm and naive Bayes are suitable for performing this subtask (i.e. candidate algorithms). In yet another example, algorithm foundation knowledge module 107 may have the information that for the subtask "regressions", the logistics regression algorithm and neural networks are suitable for performing this subtask (i.e. candidate algorithms). The domain knowledge would also include information such as the primary type of the algorithm, possible novel capabilities, parameters required and the data attributes that it can handle. Algorithm foundation knowledge module 107 is read-only in nature and cannot be over-written or modified. Preferably, algorithm foundation knowledge module 107 is not stored in a database. This is because the contents that are stored in a database can usually be modified or deleted.

As the contents of algorithm foundation knowledge module 107 are fixed, algorithm foundation knowledge module 107 is usually hardcoded.

[0048] Algorithm self-learning knowledge module 108 is read-write in nature and stores knowledge and statistics of how a particular algorithm had performed when solving a specific category of problems. In particular, algorithm self-learning knowledge module 108 stores the training statistics of each algorithm for reinforcement learning. Typical training statistics to store for reinforcement learning can be number of times an algorithm has been deployed for a particular dataset type and the accuracy score of an algorithm. Algorithm self-learning knowledge module 108 has a "self-learning" capability in that every time an algorithm is trained by algorithm orchestrator module 105, the new training statistics will be stored in algorithm self-learning knowledge module 108 so that algorithm self-learning knowledge module 108 will "grow" and "learn" over time.

[0049] Figure 2 shows the method in which system 100 automatically models a problem specified by an end-user using a user-application and orchestrates candidate algorithms to solve the problem.

[0050] In step 201, an application developer specifies the types of problems that can be solved by the user-application into universal model module 101. The types of problems can be in any field. For example, it can be in the research field, the social media field, or the field of business analytics (e.g. for computing financial performance, sales/business performance, operational performance). The problem can also be performing simulations in gene mutation or virus replication. The problem can also be in the field of gaming (e.g. predicting enemy AI movement, next game level layout).

[0051] In step 202, the application developer enters domain knowledge of the types of problems that can be solved by the user-application and the expected output into universal model module 101.

[0052] In step 203, an end-user enters the user data into the user-application which is then fed into artificial intelligence orchestrator manager module 102. The user data are the specifications of the problem to be solved.

[0053] In step 204, task manager module 103 breaks the task for solving the problem into subtasks and sends the subtasks to algorithm orchestrator module 105. These subtasks

non-exhaustively include : (1) Association; (2) Classification; (3) Clustering; (4) Combinatorial Searching; (5) Deep Web Mining; (6) Regression; (7) Imputation; (8) Sequential Decision Making; (9) Pattern Recognition; (10) Prediction; (11) NLP: topic extraction; (12) Text to numeric mapping.

[0054] In step 205, data manager module 104 checks for missing data or incomplete data, and locates missing data or imputes incomplete data, and sends the processed data to algorithm orchestrator module 105. In situations where the user data is inadequate (i.e. end-user failed to provide all the specifications of the problem or has not described the problem completely or had failed to provide or upload supporting/required data) and therefore there is "missing data", data manager module 104 can alert task manager module 103 that data is missing. Data manager module 104 can locate this missing data by "crawling" the web to locate and determine this missing data. In situations where the user data is incomplete (for example, when not all the entries for a particular column in a database are available), data manager module 104 can impute this incomplete data, for example, by using known heuristics.

[0055] In step 206, algorithm orchestrator module 105 consults algorithm foundation knowledge module 107 for the candidate algorithms for solving the subtask, and then instantiate concrete instances of these selected candidate algorithms from algorithm pool module 106. For example, algorithm orchestrator module 105 determines from algorithm foundation knowledge module 107 that the candidate algorithms for performing this particular subtask are linear regression algorithm and the logistics regression algorithm. Algorithm orchestrator module 105 then selects the linear regression algorithm and the logistics regression algorithm, and instantiates concrete instances of these algorithms via algorithm pool module 106.

[0056] In step 207, algorithm orchestrator module 105 trains each of the candidate algorithms (linear regression algorithm and the logistics regression algorithm in this example) with the processed data obtained from data manager module 104. As a result of the training, algorithm orchestrator module 105 derives training statistics pertaining to the training data for each candidate algorithm. The derived training statistics are stored in algorithm self-learning knowledge module 108. The training statistics can include recall and precision accuracy, n-fold cross validation error and confusion matrix and the like. With every training iteration or as a result of algorithms being trained, more and more training statistics will accumulate in algorithm self-learning knowledge module 108.

[0057] In step 208, algorithm orchestrator module 105 creates a Design-of-Experiment (DOE) linear combination of the selected candidate algorithms. Using the same example, if the candidate algorithms selected are linear regression algorithm and the logistics regression algorithm, the DOE may determine that one linear combination of the selected candidate algorithms (i.e. a combination of linear regression algorithm and logistics regression algorithm) has to be included .

[0058] In step 209, algorithm orchestrator module 105 builds a template POMDP model based on the candidate algorithms, the DOE linear combination of the selected candidate algorithms, and the training statistics obtained from algorithm self-learning knowledge module 108. A template POMDP model consists of 4 major components: (1) States, (2) Transition probabilities, (3) Reward functions and (4) Observation functions. The selected candidate algorithms and the DOE linear combination of the selected candidate algorithms make up the states of the template POMDP model, while the training statistics are used to create the transition probabilities, reward functions and observation functions of the template POMDP model. It is possible for the DOE linear combination of the selected candidate algorithms to be excluded from the template POMDP model, However, the advantage of template POMDP model including the DOE linear combination of the selected candidate algorithms as a state, is that the template POMDP model can find the best combination of the selected candidate algorithms so as to achieve the best accuracy (this is also known as stacking in machine learning).

[0059] An illustration of a dynamic Bayesian network representation of the template POMDP model 300 is shown in figure 3. The selected candidate algorithms (linear regression, logistics regression) and the DOE linear combination of the selected candidate algorithms (a combination of linear regression and logistics regression) form states 301 of template POMDP model 300. Auxiliary states 302 indicating, “start”, “good” and “bad” serve as confirmation and terminal states.

[0060] Template POMDP model 300 comprises first layer transition probabilities t_{ij} (between auxiliary state 302 "start" and states 301) and second layer transition probabilities t'_{ij} (between states 301 and auxiliary states 302 "Good" and "Bad"). First layer transition probabilities t_{ij} can be the Probably Approximately Correct Markov Decision Process (PAC-MDP) statistics of the selected candidate algorithms obtained from previous runs, and can be

retrieved from algorithm self-learning knowledge module 108. PAC-MDP is an algorithm that helps to choose the probability distribution among a set of possibilities such that the long run regret or cost is minimized. It is a distribution rather than a crisp discrete point because there are many possibilities, some of which might not have been tested before. Therefore, it cannot discount those and must maintain a probability distribution to account for these unknowns. So in the case of the template POMDP model, the PAC-MDP algorithm based on statistics of previous runs will set first layer transition probabilities t_{ij} to reflect the best possible outcome. In figure 3, the first layer transition probabilities t_{ij} are t_{11}, t_{12} and t_{13}. Second layer transition probabilities t'_{ij} can be training errors, precision and recall metrics obtained by algorithm orchestrator module 105 creating concrete instances of the selected candidate algorithms and running them against the processed data and past knowledge. In figure 3, the second layer transition probabilities t'_{ij} are t'_{11}, t'_{12}, t'_{21}, t'_{22}, t'_{31} and t'_{32}.

[0061] Reward functions r_{ij} and r'_{ij} are the functions of the priority of the subtask, real time constraints and the accuracy desired. The reward function is dependent on which state the agent is in. Since the “good” and “bad” states serve as confirmation actions, landing in those states should have a wide enough reward distinction, typically an order of magnitude, e.g. 10 and -1 respectively. Empirical testing and experiments are usually done to calibrate the reward function. In figure 3, the reward functions are r_{11}, r_{12}, r_{13}, r'_{11}, r'_{12}, r'_{21}, r'_{22}, r'_{31} and r'_{32}.

[0062] Observation functions follow an underlying distribution that provides feedback for the template POMDP model. The feedback could be an accuracy level for each of the selected candidate algorithms. This is important because many algorithms use probability bounds and confidence intervals which observations can be considered “noisy”. Transition probabilities, reward functions and observation functions can be self-learned over time as algorithm orchestrator module 105 models more and more problems.

[0063] Referring to template POMDP model 300 in figure 3, there is a t_{11} probability to transit from auxiliary state 302 “start” to state 301 “linear regression algorithm”, and if it does, it will receive a reward of r_{11}. By transiting from one state to another, the observation function would be the state 301 that the template POMDP ended up in. Therefore, there is a t_{11} probability to transit from auxiliary state 302 “start” to state 301 “linear regression algorithm”, and if it does, it will observe that it is in state “linear regression algorithm”.

[0064] In essence, template POMDP model 300 represents the “structure” and the “weights” of the selected candidate algorithms (and the DOE linear combination of the selected candidate algorithms). The "structure" of the algorithms is represented by states 301 while the "weights" or probability of the algorithms are implicitly represented by first layer transition probabilities t_{ij} into each of states 301 and second layer transition probabilities t'_{ij} into each of auxiliary states 302 "Good" and "Bad".

[0065] In step 210, algorithm orchestrator module 105 parametrizes the template POMDP model into POMDP model parameters. This process allows the template POMDP model to be represented purely by numerals. An illustration of parametrizing template POMDP model is shown in figure 4.

Figure 4 provides exemplary values for first layer transition probabilities t_{ij}:

i) t_{11} = [0.2, 0.5], which indicates the range of probabilities t_{11} can have i.e. from 0.2 to 0.5, t_{11} being the probability of algorithm orchestrator module 105 using the linear regression algorithm;
ii) t_{12} = [0.3, 0.6], which indicates the range of probabilities t_{12} can have i.e. from 0.3 to 0.6, t_{12} being the probability of algorithm orchestrator module 105 using the logistics regression algorithm;
iii) t_{13} = [0.1, 0.8], which indicates the range of probabilities t_{13} can have i.e. from 0.1 to 0.8; t_{13} being the probability of algorithm orchestrator module 105 using a combination of the linear regression algorithm and the logistics regression algorithm;

where the sum of t_{11} and t_{12} and t_{13} = 1.0.

Figure 4 provides exemplary values for second layer transition probabilities t'_{ij} :

i) t'_{11} = [0.2, 0.5], which indicates the range of probabilities t'_{11} can have i.e. from 0.2 to 0.5, t'_{11} being the probability of the linear regression algorithm transitioning to the "Good" state;
ii) t'_{12} = [0.3, 0.8], which indicates the range of probabilities t'_{12} can have i.e. from 0.3 to 0.8, t'_{12} being the probability of the linear regression algorithm transitioning to the "Bad" state;

where the sum of t'_{11} and t'_{12} = 1.0;

iii) t'_{21} = [0.1, 0.8], which indicates the range of probabilities t'_{21} can have i.e. from 0.1 to 0.8, t'_{21} being the probability of the logistics regression algorithm transitioning to the "Good" state;

iv) t'_{22} = [0.2, 0.9], which indicates the range of probabilities t'_{22} can have i.e. from 0.2 to 0.9, t'_{22} being the probability of the logistics regression algorithm transitioning to the "Bad" state;
where the sum of t'_{21} and t'_{22} = 1.0;

v) t'_{31} = [0.2, 0.65], which indicates the range of probabilities t'_{31} can have i.e. from 0.2 to 0.65, t'_{31} being the probability of a combination of the linear regression algorithm and the logistics regression algorithm transitioning to the "Good" state;

vi) t'_{32} = [0.3, 0.8], which indicates the range of probabilities t'_{32} can have i.e. 0.3 to 0.8; t'_{32} being the probability of a combination of the linear regression algorithm and the logistics regression algorithm transitioning to the "Bad" state;
where the sum of t'_{31} and t'_{32} = 1.0.

[0066] Therefore, exemplary POMDP model parameters for the template POMDP model 300 are :- t_{11} = [0.2, 0.5], t_{12} = [0.3, 0.6], t_{13} = [0.1, 0.8], t'_{11} = [0.2, 0.5], t'_{12} = [0.3, 0.8], t'_{21} = [0.1, 0.8], t'_{22} = [0.2, 0.9], t'_{31} = [0.2, 0.65], t'_{32} = [0.3, 0.8].

[0067] The values provided for the transition probabilities are "exemplary" only. Typically, from a cold start, without any benefit of past statistics of the selected candidate algorithms obtained from previous runs, the range of probabilities for the transition probabilities would be [0.0, 1.0]. But over time, as more training statistics are gathered, the self-learning nature of algorithm self-learning knowledge module 108 may determine that for certain transitions, the probability is within a more confined range such as [0.2, 0.5].

[0068] In step 211, algorithm orchestrator module 105 uses a genetic algorithm to optimize the POMDP model parameters (t_{11}, t_{12}, t_{13}, t'_{11}, t'_{12}, t'_{21}, t'_{22}, t'_{31}, t'_{32}). Figure 5 shows the steps taken by algorithm orchestrator module 105 in using a genetic algorithm to optimize the POMDP model parameters.

[0069] In step 501, algorithm orchestrator module 105 uses the genetic algorithm to initialize the population of candidates (sets of mutated POMDP model parameters) in the first

epoch. These candidates are culled from the POMDP model parameters (t_{11}, t_{12}, t_{13}, t'_{11}, t'_{12}, t'_{21}, t'_{22}, t'_{31}, t'_{32}). For example, the genetic algorithm may randomly choose values for:-

t_{11} to be 0.3;

t_{12} to be 0.5;

t_{13} to be 0.2;

as long as the conditions of t_{11} being within the range [0.2 and 0.5], t_{12} being within the range [0.3 and 0.6], t_{13} being within the range [0.1 and 0.8], and the sum of t_{11} and t_{12} and t_{13} = 1.0, are satisfied.

The genetic algorithm may randomly choose values for:-

t'_{11} to be 0.3;

t'_{12} to be 0.7;

as long as the conditions of t'_{11} being within the range [0.2 and 0.5], t'_{12} being within the range [0.3 and 0.8] and the sum of t'_{11} and t'_{12} = 1.0, are satisfied.

The genetic algorithm may randomly choose values for:-

t'_{21} to be 0.5;

t'_{22} to be 0.5;

as long as the conditions of t'_{21} being within the range [0.1 and 0.8], t'_{22} being within the range [0.2 and 0.9] and the sum of t'_{21} and t'_{22} = 1.0, are satisfied.

The genetic algorithm may randomly choose values for:-

t'_{31} to be 0.6;

t'_{32} to be 0.4;

as long as the conditions of t'_{31} being within the range [0.2 and 0.65], t'_{32} being within the range [0.3 and 0.8] and the sum of t'_{31} and t'_{32} = 1.0, are satisfied. This forms the first set of mutated POMDP model parameters. The other sets of mutated POMDP model parameters are also created in this fashion and the sets of mutated POMDP model parameters form the initial population of candidates.

[0070] In step 502, for each candidate (or set of mutated POMDP model parameters) in the population, algorithm orchestrator module 105 instantiates a candidate POMDP model and solves the resulting candidate POMDP model. The outcome of solving each candidate POMDP model is a policy. A policy is a set of instructions for telling an “agent” executing the policy what to do in what situation. Therefore, in the context of this invention, the policy is a “recipe” for executing the selected candidate algorithms.

[0071] In step 503, algorithm orchestrator module 105 runs each policy via a simulator or evaluator to determine the value of the policy. The value of the policy determines the fitness of each candidate POMDP model.

[0072] In step 504, algorithm orchestrator module 105 ranks the fitness of the sets of the POMDP model parameters which resulted in the candidate POMDP models using a sorting algorithm. Preferably, the sorting algorithm can be non-dominated sorting. Non-dominated sorting allows algorithm orchestrator module 105 to compute the set of Pareto optimal solutions.

[0073] In step 505, if the termination criterion is not met (which can be a fixed number of iterations or it can be a threshold for the fitness value), algorithm orchestrator module 105 performs the "mutation" and crossover operators to produce new sets of mutated POMDP model parameters which form the population of candidates for the next epoch. The "mutation' is usually performed on higher ranked (and therefore better fitness) mutated POMDP model parameters. The crossover operator may also partition the candidates or sets of mutated POMDP model parameters and swaps the partitions to generate the offspring. The mutation operator may flip or change a parameter in a set of mutated POMDP model parameters to generate the offspring. Depending on the elitism policy, either the top half of the population will get to participate in the crossover or only the set of mutated POMDP model parameters which resulted in the best ranked candidate POMDP model.

[0074] At the end of the genetic algorithm run (when the fixed number of iterations has ran or the threshold for the fitness value has been reached i.e. the termination criterion has been met), in step 506, algorithm orchestrator module 105 extracts the best performing individual, i.e. the set of mutated POMDP model parameters which resulted in the best ranked candidate POMDP model in the population. This set of mutated POMDP model parameters would be the optimized POMDP model parameters.

[0075] Referring back to figure 2, in step 212, algorithm orchestrator module 105 instantiates a final POMDP model with the optimized POMDP model parameters and solves this final POMDP model to obtain a policy.

[0076] In step 213, the policy is ran and a sequence and weight of executing the selected candidate algorithms is obtained. "Sequence" being the order in which the selected

candidate algorithms are executed while "weight" being the amount in which the selected candidate algorithms are executed.

[0077] In step 214, the selected candidate algorithms are executed in accordance with the "sequence" and "weight" to obtain an output. The output will be the solution to the problem. Or in situations where the task has been broken into multiple subtasks, the output will be a partial solution to the problem.

[0078] Therefore, the invention discloses a novel way of parametrizing the template POMDP model into POMDP model parameters (which are essentially a "string of numbers), and then using the genetic algorithm to "mutate" the POMDP model parameters to form sets of mutated POMDP model parameters which form the population of candidates. At each epoch, the sets of mutated POMDP model parameters are evaluated by converting these string of numbers into candidate POMDP models, solving them (to arrive at the policies), and execute the policies to obtain the value of each policy. The genetic algorithm then ranks these mutated POMDP model parameters according to the value of their policy, from highest to the lowest. “Higher ranked” mutated POMDP model parameters will have a higher chance to be mutated during the next epoch, while lower ranked mutated POMDP model parameters will have a lower chance to be mutated during the next epoch. At the end of the genetic algorithm run (when the fixed number of iterations has ran or the threshold for the fitness value has been reached), the set of mutated POMDP model parameters which resulted in the best ranked candidate POMDP model in the population would be the optimized POMDP model parameters. From the optimized POMDP model parameters, the solution to the problem can be derived.

[0079] The invention therefore teaches a computer implemented method and system that is capable of formulating a problem into a template POMDP model automatically with a computer and without human input (steps 204, 205, 206, 207, 208 and 209 in figure 2). The invention also teaches a computer implemented method and system that is capable of solving the template POMDP model automatically with a computer and without human input (steps 210, 211, 212, 213 and 214 in figure 2). This is opposed to the current human-driven approach, which is mostly an iterative trial and error process of defining and changing the parameters of the POMDP model, before the ideal end-results can be identified.

[0080] The technical advantage of the system and method as described is that it is autonomous in formulating a problem into a template POMDP model and finding the ideal

solution to the problem. Therefore, this system can be used by application developers with minimal machine learning knowledge and skills. The input required by the application developers would only be to specify the types of problems the user-application can solve (step 201 in figure 2) as well as provide the domain knowledge of the types of problems and expected output (step 202 in figure 2). Furthermore, one can appreciate that with the automation of formulating the problem into a template POMDP model, and subsequently finding the ideal solution to the POMDP model, the technical effect contributed by the invention is that problems can be solved faster and more efficiently. This system is also robust to solve different types of mathematical problems (e.g. business analytics, simulations for gene mutation or virus replication etc).

[0081] A single processor or a plurality of processors can implement house universal model module 101, artificial intelligence orchestrator manager module 102, task manager module 103, data manager module 104, algorithm orchestrator module 105, algorithm pool module 106, algorithm foundation knowledge module 107 and algorithm self-learning knowledge module 108. Alternatively, a server can have one or more processors or databases for implementing universal model module 101, artificial intelligence orchestrator manager module 102, task manager module 103, data manager module 104, algorithm orchestrator module 105, algorithm pool module 106, algorithm foundation knowledge module 107 and algorithm self-learning knowledge module 108. Alternatively, universal model module 101, artificial intelligence orchestrator manager module 102, task manager module 103, data manager module 104, algorithm orchestrator module 105, algorithm pool module 106, algorithm foundation knowledge module 107 and algorithm self-learning knowledge module 108 can be implemented by different servers. One skilled in the art will appreciate that other variations are possible.

[0082] In the application, unless specified otherwise, the terms "comprising", "comprise", and grammatical variants thereof, intended to represent "open" or "inclusive" language such that they include recited elements but also permit inclusion of additional, non-explicitly recited elements.

[0083] It will be apparent that various other modifications and adaptations of the application will be apparent to the person skilled in the art after reading the foregoing disclosure without departing from the spirit and scope of the application and it is intended that all such modifications and adaptations come within the scope of the appended claims.

CLAIMS

1. A computer implemented method for automatically modelling a problem and orchestrating algorithms to obtain a solution to the problem, the method comprising the steps of :-

establishing with an artificial intelligence orchestrator manager module, that a task has to be performed to solve the problem;

deriving with a task manager module, at least one subtask from the task;

selecting with an algorithm orchestrator module, a plurality of candidate algorithms based on the at least one subtask;

training with the algorithm orchestrator module, the plurality of selected candidate algorithms to obtain training statistics;

building with the algorithm orchestrator module, a template POMDP (partially observable markov decision process) model based on the plurality of selected candidate algorithms and the training statistics;

parametrizing with the algorithm orchestrator module, the template POMDP model into POMDP model parameters; and

implementing with the algorithm orchestrator module, a genetic algorithm to optimize the POMDP model parameters, wherein the solution to the problem is derived from the optimized POMDP model parameters.

2. The computer implemented method of claim 1 further comprising the step of creating with the algorithm orchestrator module, at least one design-of-experiment linear combination of the plurality of selected candidate algorithms; and wherein building the template POMDP model will be based on the plurality of selected candidate algorithms, the training statistics and the at least one design-of-experiment linear combination of the plurality of selected candidate algorithms.

3. The computer implemented method of claim 2 wherein the template POMDP model comprises states, transition probabilities, reward functions and observation functions; and wherein the plurality of selected candidate algorithms and the at least one design-of-experiment linear combination of the plurality of selected candidate algorithms form the states of the template POMDP model, and the transition probabilities, the

reward functions and the observation functions of the template POMDP model are derived from the training statistics.

4. The computer implemented method of any one of the preceding claims wherein the step of implementing with the algorithm orchestrator module, a genetic algorithm to optimize the POMDP model parameters comprises the steps of :-

    (a) mutating the POMDP model parameters with the genetic algorithm to form sets of mutated POMDP model parameters;
    (b) instantiating a candidate POMDP model for each set of mutated POMDP model parameters and solve the POMDP candidate model to obtain a policy;
    (c) determining the value of the policy for each set of mutated POMDP model parameters;
    (d) using a sorting algorithm to rank the sets of mutated POMDP model parameters based on the value of the policy, to obtain a highest ranked set of mutated POMDP model parameters;
    (e) repeating steps (a), (b), (c) and (d) if a termination criterion is not met; and
    (f) returning the highest ranked set of mutated POMDP model parameters as the optimized POMDP model parameters if the termination criterion is met.

5. The computer implemented method of claim 4 wherein the termination criterion is when a fixed number of iterations has ran or when a threshold value for the policy has been reached.

6. The computer implemented method of claim 4 or claim 5 wherein the sorting algorithm is non-dominated sorting.

7. The computer implemented method of any one of the preceding claims further comprising the step of receiving with a universal model module, the domain knowledge and types of problems that can be solved by a user-application.

8. The computer implemented method of claim 7 further comprising the step of receiving with the artificial intelligence orchestrator manager module, user data from the user-application, the user data specifying the problem to be solved.

9. The computer implemented method of claim 8 further comprising the steps of checking and ensuring the data integrity of the user data with a data manager module, and providing the data integral user data to algorithm orchestrator module for training the plurality of selected candidate algorithms.

10. The computer implemented method of any one of the preceding claims wherein the step of selecting with an algorithm orchestrator module, a plurality of candidate algorithms based on the at least one subtask comprises the steps of :-

consulting an algorithm foundation knowledge module and selecting the plurality of candidate algorithms that correspond to the at least one subtask; and

instantiating concrete instances of the plurality of selected candidate algorithms from an algorithm pool module.

11. The computer implemented method of any one of the preceding claims further comprising the step of storing the training statistics in an algorithm self-learning knowledge module.

12. The computer implemented method of any one of the preceding claims wherein the plurality of selected candidate algorithms are any combination of the following: POMDP, genetic algorithm, artificial neural networks, natural language processing, fuzzy logic, Apiori, k-means clustering, adaptive boosting, support vector machine, naive Bayes and logistics regression.

13. A computer program product embodied on a computer readable medium containing instructions to perform the computer implemented method of any one of the preceding claims.

14. A system for automatically modelling a problem and orchestrating algorithms to obtain a solution to the problem, the system comprising :-

at least one processor programmed to implement an artificial intelligence orchestrator manager module, a task manager module and an algorithm orchestrator module;

wherein the artificial intelligence orchestrator manager module is configured to establish that a task has to be performed to solve the problem;

wherein the task manager module is configured to derive at least one subtask from the task; and

wherein the algorithm orchestrator module is configured to :-

select a plurality of candidate algorithms based on the at least one subtask;

train the plurality of selected candidate algorithms to obtain training statistics;

build a template POMDP (partially observable markov decision process) model based on the plurality of selected candidate algorithms and the training statistics;

parametrize the template POMDP model into POMDP model parameters; and

implement a genetic algorithm to optimize the POMDP model parameters, wherein the solution to the problem is derived from the optimized POMDP model parameters.

15. The system of claim 14 wherein the algorithm orchestrator module is further configured to :-

create at least one design-of-experiment linear combination of the plurality of selected candidate algorithms; and

build the template POMDP model based on the plurality of selected candidate algorithms, the training statistics and the at least one design-of-experiment linear combination of the plurality of selected candidate algorithms.

16. The system of claim 15 wherein the template POMDP model comprises states, transition probabilities, reward functions and observation functions; and wherein the plurality of selected candidate algorithms and the at least one design-of-experiment linear combination of the plurality of selected candidate algorithms form the states of the template POMDP model, and the transition probabilities, the reward functions, and the observation functions of the template POMDP model are derived from the training statistics.

17. The system of any one of claims 14 to 16 wherein the algorithm orchestrator module is further configured to:-

(a) mutate the POMDP model parameters with the genetic algorithm to form sets of mutated POMDP model parameters;

(b) instantiate a candidate POMDP model for each set of mutated POMDP model parameters and solve the POMDP candidate model to obtain a policy;
(c) determine the value of the policy for each set of mutated POMDP model parameters;
(d) use a sorting algorithm to rank the sets of mutated POMDP model parameters based on the value of the policy, to obtain a highest ranked set of mutated POMDP model parameters;
(e) repeat steps (a), (b), (c) and (d) if a termination criterion is not met; and
(f) return the highest ranked set of mutated POMDP model parameters as the optimized POMDP model parameters if the termination criterion is met.

18. The system of claim 17 wherein the termination criterion is when a fixed number of iterations has ran or when a threshold value for the policy has been reached.

19. The system of claim 17 or claim 18 wherein the sorting algorithm is non-dominated sorting.

20. The system of any one of claims 14 to 19 wherein the at least one processor is further programmed to implement a universal model module, wherein the universal model module is configured to receive domain knowledge and types of problems that can be solved by a user-application.

21. The system of claim 20 wherein the artificial intelligence orchestrator manager module is further configured to receive user data from the user-application, the user data specifying the problem to be solved.

22. The system of claim 21 wherein the at least one processor is further programmed to implement a data manager module to check and ensure the data integrity of the user data, and provide the data integral user data to algorithm orchestrator module for training the plurality of selected candidate algorithms.

23. The system of any one of claims 14 to 22 wherein the at least one processor is further programmed to implement an algorithm foundation knowledge module and an

algorithm pool module, and the algorithm orchestrator module selects the plurality of candidate algorithms based on the at least one subtask by :-

consulting the algorithm foundation knowledge module and selecting the plurality of candidate algorithms that correspond to the at least one subtask; and

instantiating concrete instances of the plurality of selected candidate algorithms from the algorithm pool module.

24. The system of any one of claims 14 to 23 wherein the at least one processor is further programmed to implement an algorithm self-learning knowledge module to store the training statistics.

25. The system of any one of claims 14 to 24 wherein the plurality of selected candidate algorithms are any combination of the following: POMDP, genetic algorithm, artificial neural networks, natural language processing, fuzzy logic, Apiori, k-means clustering, adaptive boosting, support vector machine, naive Bayes and logistics regression.