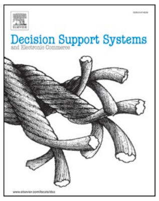
Decision Support Systems 144 (2021) 113490

Contents lists available at ScienceDirect

Decision Support Systems

journal homepage: www.elsevier.com/locate/dss



A strategic decision-making architecture toward hybrid teams for dynamic competitive problems

Alparslan Emrah Bayrak [a,*], Christopher McComb [b], Jonathan Cagan [c], Kenneth Kotovsky [d]

[a] *School of Systems and Enterprises, Stevens Institute of Technology, Hoboken, NJ 07030, United States of America*
[b] *School of Engineering Design, Technology, and Professional Programs, The Pennsylvania State University, University Park, PA 16802, United States of America*
[c] *Department of Mechanical Engineering, Carnegie Mellon University, Pittsburgh, PA 15213, United States of America*
[d] *Department of Psychology, Carnegie Mellon University, Pittsburgh, PA 15213, United States of America*

A R T I C L E I N F O

Keywords:
Complex systems
Feedback control
Differential games
Strategic decisions
Task partitioning DECSUP-D-20-00719R1

A B S T R A C T

Advances in artificial intelligence create new opportunities for computers to support humans as peers in hybrid teams in several complex problem-solving situations. This paper proposes a decision-making architecture for adaptively informing decisions in human-computer collaboration for large-scale competitive problems under dynamic environments. The proposed architecture integrates methods from sequence learning, model predictive control, and game theory. Computers in this architecture learn objectives and strategies from experimental data to support humans with strategic decisions while operational decisions are made by humans. The paper also presents data-driven methods for partitioning tasks among a team of computers in this architecture. The generalized methodology is illustrated on the real-time strategy game Starcraft II. The results from this application show that low-performing players can benefit from the game-theoretic decision support whereas this support can be overly conservative for high-performing players. The proposed approach provides safe though suboptimal suggestions particularly against an opponent with an unknown level of expertise. The results further show that problem solution with a team of computers based on non-intuitive task partitioning significantly improves the quality of decisions compared to an all-in-one solution with a single computer.

1. Introduction

Human-computer collaboration has recently gained more attention for its potential to extend the capability of human decision-makers to address increasingly complex problems common in engineering system management or military operations. Effective team collaboration is essential, especially for large-scale problems that go beyond the expertise and cognitive capacity of a single decision-maker. Using automated systems to support human decision-making dates back to the 1950s with the Fitts List which has identified humans and computers to have unique capabilities that can complement each other in teamwork [1]. Since then, several decision support systems have been developed using similar principles to allocate work between humans and computers [2]. With the recent advances in machine learning, computers can now extract useful knowledge from big data, which is a capability that earlier automated systems lacked. Hence, there are new opportunities to augment humans' intuition that is generally required to solve open-ended or more ill-defined problems with computers equipped with state-of-the-art artificial intelligence (AI), that can outperform human decision-making in well-structured problems such as chess or Go [3]. An extensive review of how data-driven intelligent systems are used to improve managerial decisions in business can be found in [4].

We posit that a properly designed *hybrid team* that leverages these complimentary capabilities can achieve more robust decisions than humans or computers alone. Previous studies have used computer automation as a decision support for humans in well-defined operational problems and provided guidelines for developing automation for this purpose [5]. Contrary to this common approach where computer agents perform well-defined repetitive tasks [6], this study presents a new perspective, not yet studied in the literature, that explores the possibility of using computers for higher-level strategic decision-making in collaboration with humans. In this perspective, a computer could take feedback from the human partner and take a proactive role to support human decision-making and allow exploring novel solutions to complex real-world problems.

This article focuses on the solution of dynamic competitive problems

* Corresponding author.
E-mail addresses: ebayrak@stevens.edu (A.E. Bayrak), mccomb@psu.edu (C. McComb), cagan@cmu.edu (J. Cagan), kotovsky@cmu.edu (K. Kotovsky).

https://doi.org/10.1016/j.dss.2020.113490
Received 3 June 2020; Received in revised form 29 December 2020; Accepted 30 December 2020
Available online 4 January 2021

(DCPs) using hybrid teams with a particular focus on problems with high complexity. Examples to such problems include operations management of engineering systems throughout their lifecycle under dynamic market competition and adversarial decision-making in military squads in the theater of war [7]. These problems are characterized by two specific attributes: 1) continuous decision-making over time under dynamically changing environments and 2) a competition against an opponent. Adaptation to the changes in the problem settings by effectively countering the opponent strategy is critical for successful decisions. These problems are made even more challenging by several sources of complexity, including problem uncertainty, lack of clearly defined objectives, high-dimensional decision spaces, and time-varying problem characteristics. Absence of a mathematical model that captures all these factors and the curse of high dimensionality make problems of this type difficult for both human and AI.

The current work addresses these challenges by using a bi-level problem representation that separates short-term (operational) and long-term (strategic) decisions. Representing complex problems with two or more levels allows mathematically formulating and analyzing each problem with separate decision-making processes, as discussed in operations research and systems engineering literature [8,9].

This paper presents an adaptive decision-making process as a foundational development for computers in hybrid teams by integrating methods from sequence learning, model predictive control and game theory. Solving a game theoretic problem for a dynamic system using control methods is referred to as a *differential game* in the literature [10]. Model predictive control [11] used in this paper uses a set of differential equations to predict the future system state and optimizes control actions for a future prediction horizon based on a given system objective. This paper trains a sequence learning model from prior data as a system objective function, referred to as a reward model. This reward model evaluates the quality of control decisions in a dynamic system.

The present work does not model human response by making an assumption that a human member in the hybrid team follows all the decisions suggested by the computers. Verifying the efficacy of human-computer collaboration with human participants in the loop is left for future work. Approaches in this paper are presented at a level of generalization that facilitates their application to a wide range of domains. This article builds upon the earlier work in [12].

1.1. *Background on differential games*

Adaptive feedback mechanisms for dynamical systems and equilibrium solutions for conflicting situations have been well studied in control [13] and game [14] theories, respectively. The analysis of competition in dynamic systems requires merging these two theories, which led to the emergence of differential games [10]. Competitions in differential games are quite distinct from the classical game theory since dynamic systems continuously evolve over time without waiting for a decision from any of the players, whereas in the classical game theory, a turn-based format is commonly assumed. Originally motivated by pursuit-evasion problems, differential games have been applied to a variety of problems including military systems [10], economics and management [15].

Normative studies have developed analytical solutions to relatively simplified systems to understand the factors that impact competitive decision-making in dynamic environments. These studies commonly use linear-quadratic regulators [16] and dynamic programming [17] to find equilibrium solutions to differential games. Applicability of these approaches to practical problems is limited. Linear-quadratic regulator-based approaches make simplistic assumptions in their dynamic system and reward models which may not be valid for many real-world problems, and dynamic programming is not scalable to systems with a large number of decision variables.

1.2. *Background on human-computer collaboration*

Exploring the possibility and the potential of human-computer teaming for problem solving has been an area of interest in the recent literature. Studies have shown that it is possible to emulate the human problem-solving process with computers in configuration design problems [18], and also that human team members may be replaced with computer agents for the coordination of an unmanned aerial system [19]. These demonstrate the feasibility of human-computer partnering. Literature on human-robot and human-computer interaction have studied how humans interact with robots with low levels of autonomy [20,21] and identified important human factors that impact the outcome of the collaboration [6]. Several issues including human trust [22], issues related to attention deficit [23] and automation transparency [24] along with their impact on human decision-making have been studied in the literature. Existing literature mostly focuses on automation without any particular emphasis on learning or adaptation capability that is enabled by machine learning methods. A systematic method to develop computer agents that can adaptively support and collaborate with humans in a hybrid team under dynamic competition that involve short-term and long-term decisions remains unexplored, as do methods to perform a division of labor in such hybrid teams.

1.3. *Contributions*

The primary contribution of this paper is an adaptive decision-making architecture that can facilitate human-computer collaboration for DCPs based on the integration of methods from sequence learning, model predictive control, and game theory. This architecture represents the decisions for complex systems in multiple timescales, where the computer agents in a hybrid team learn objectives and strategies from prior empirical data. These computer agents provide suggestions for strategic decisions to a potential human partner engaged with long-term decision-making. The proposed approach uses empirical data without making assumptions about a particular problem formulation and particularly addresses the complexity of real-world problems. Computational results from the proposed methods are demonstrated with a symmetric zero-sum dynamic competition defined on the real-time strategy game Starcraft II [25].

Fig. 1 shows an overview of the integrated modeling approach and the two-phase decision-making process introduced in this paper. Prior *empirical gameplay data* (Fig. 1a) is processed based on a *bi-level problem representation* (Fig. 1b). The proposed approach identifies a variety of strategies used by humans via *clustering* (Fig. 1c) and learns their objectives from this data with a *reward model* (Fig. 1d). Then, a two-phase decision-making architecture starting with an *opening phase* (Fig. 1e) followed by a *differential game-based adaptive phase* (Fig. 1f) is introduced. Each element in this figure is described in subsequent sections. Section 2 describes the important characteristics of DCPs and the mathematical modeling approach; Section 3 introduces the proposed human-computer collaboration architecture and the problem solution strategy; Section 4 presents the analysis of partitioned problem solving with computer teams; Section 5 discusses the outcomes and provides important take-aways; and Section 6 states the conclusions, limitations and directions for further research.

2. Model of dynamic competitive problems

DCPs require making adaptive decisions based on the dynamics of the overall system while simultaneously accounting for the potential decisions of the opponent. This paper focuses on the solution of

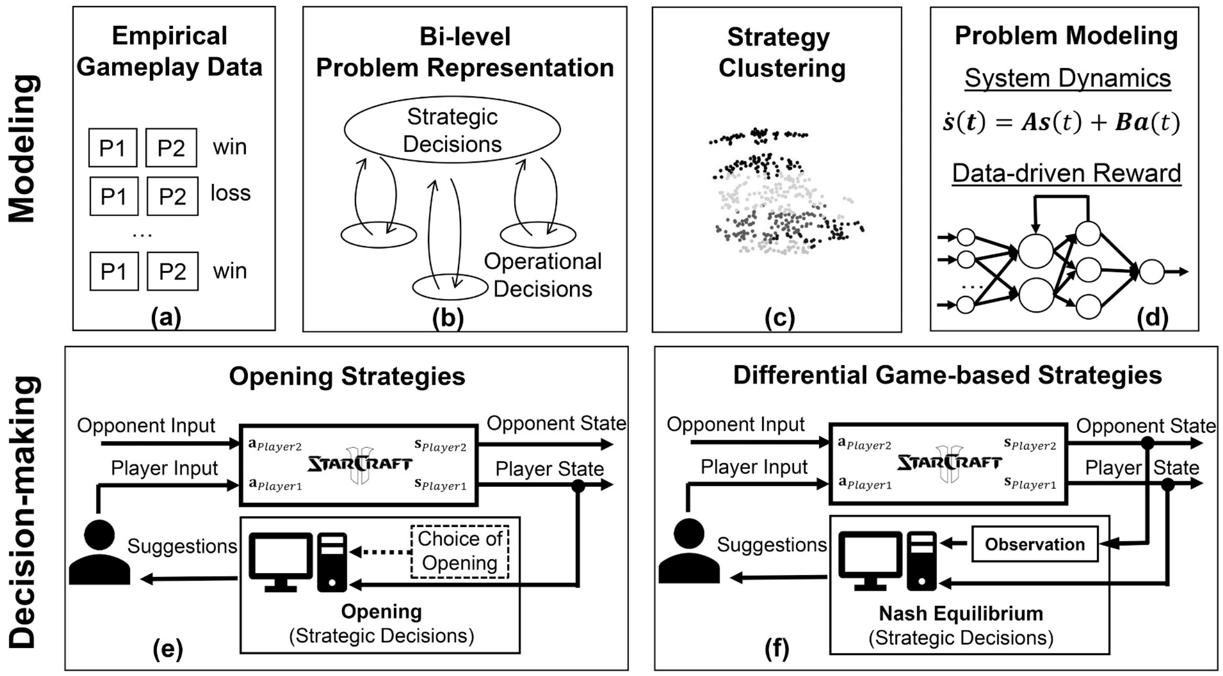


Fig. 1. Overview of the integrated modeling approach (a–d) and the two phases of the proposed decision-making architecture (e–f).

problems with high complexity which can partially be characterized by 1) a large number of interrelated decisions 2) with non-linear relationships 3) in a dynamic system that evolves over time 4) with adaptive feedback loops due to the competitive nature [26,27].

DCPs differ from the applications that use the classical game theory since the decisions have to be made continuously over time by all parties. Therefore, discretizing decisions to represent the problem with decision-trees as used in chess or Go [28] is not applicable to these systems. Also, DCPs consider longer-term objectives to search for an equilibrium solution while classical applications of competitive problems generally restrict the analysis horizon (i.e., the depth of search) based on the computational limits [29].

This paper uses the game Starcraft II as an application platform to demonstrate the proposed methods on a complex DCP, while introducing the methods in such a way that they may be generalized to a wide range of application domains. Starcraft II is a real-time strategy game with a goal of defeating an opponent by collecting and managing resources to build a base and create an army composed of various unit types [25]. The game features three races, namely, *terran*, *protoss* and *zerg*, which have unique characteristics. Each race can construct different buildings which enable training different types of units with unique capabilities. These capabilities can be enhanced by upgrades. As there is no single strategy to defeat all opponents, managing harvested resources to counter the opponent strategy is critical to win the game. For simplicity, this paper examines only terran-vs-terran matches to model a symmetric zero-sum dynamic competition in Starcraft II. Symmetry comes from the fact that both players start with the same resources and have the same capabilities of the terran race, and zero-sum game is due to the adversarial nature of Starcraft II where one player wins by damaging the other player's base.

Starcraft II is comparable in complexity to many real-world problems. The game does not define an explicit objective besides the end-goal of defeating the opponent. In addition to the direct analogy to military applications, this problem characteristic is also analogous to design problems where mapping the overall goal of product profit to individual design decisions cannot be made explicitly. There are many decisions that make the problem size intractable, including the decisions on managing resources, units, buildings and upgrades. The problem size also varies based on the decisions made in the game. For instance, the decision on how many units of a particular type to train determines the size of the state space to control these units. Such a complex problem as that defined by Starcraft II provides a suitable research platform to study design and management problems that cannot be completely modeled and solved by existing optimization methods.

A common approach in the AI literature to address the problem in Starcraft is to develop different AI methods for smaller subproblems defined out of the overall problem manually (such as collecting resources starting with a given number of workers or finding and defeating opponent units with a given number of player units) [30]. These methods fail to solve the overall problem without an integrated approach that brings the solutions of the subproblems together by accounting for the interconnections among them. State-of-the-art AI methods for Starcraft perform at the level of an amateur human player but were unable to defeat a human expert due to the problem complexity [31], until recently [32].

Instead of replacing humans with computers, the present work develops a decision-making process for computers to collaborate with humans. This study distinguishes *strategic* and *operational* decisions based on the timescale of decision-making, similar to the existing temporal modeling approaches in operations research and systems engineering [8,9]. This distinction simplifies the mathematical formulation since strategic decisions with long-term objectives determine the problem space for operational decisions with near-term objectives. This paper focuses on the human-computer collaboration for strategic decision-making assuming that the operational decisions are left to humans in hybrid teams. The rest of this section describes the mathematical representation that models the system dynamics at strategic timescales and the time-series model for reward trained with existing game play data.

2.1. System dynamics at strategic timescale

The separation between the strategic and operational decisions have been commonly discussed in management and engineering literature

[33]. Some studies classify decisions into three categories by including a tactical level between strategic and operational decisions based on the timescale that these decisions take effect [8,9]. Depending on the application, the decision-making timescale could be separated into years, months and days [9] or months, days and minutes [8] for these three levels of decision-making. The present study excludes the tactical level for simplicity. In engineering systems broadly, investments that expand the capacity of a system (e.g., new warehouses in a distribution network, new vehicles in a car share system) can be examples of strategic decisions, whereas management of existing resources (e.g., scheduling deliveries in a distribution network, vehicle routing in a car share system) can be operational decisions for the same context. In Starcraft II specifically, this paper follows the commonly used separation of macro- and micro-level decisions to distinguish strategic and operational decisions [25]. Macro-level decisions involve big-picture economy management (i.e., resource harvesting and investment-related decisions) and tactics (i.e., taking a defensive or offensive stance) and are referred to as strategic decisions in this paper. Micro-level decisions involve moving units around the map, assigning individual units to specific enemy targets to attack and determining locations to site the buildings, and are referred to as operational decisions. While alternative classifications of decisions are possible, the methods presented in the paper do not depend on this particular definition of strategic and operational decisions.

The boundary between strategic and operation decisions is generally subjective and requires some intuition about the system [34]. The subjectivity in problem representation also exists in many engineering applications. For instance, an optimal control or operations management problem can be represented with decisions on policy parameters in a network representation or with time-dependent variables in a state-space model.

System dynamics at strategic timescale can be modeled with a state-space representation that relates action and state variables using a set of differential equations. In such a state-space representation, the action variables correspond to the independent decisions made to control the dynamic system, and the state variables correspond to the dependent variables, the collection of which fully describe the system and its response. Strategic decisions, i.e., action variables in Starcraft II, can be classified into four categories corresponding to allocating workers to gather resources, creating new units and buildings, upgrading unit capabilities and sending units to the opponent territory to attack (or to keep at the player territory for defense) at any given time. Similarly, corresponding state variables represent the quantity of resources available in the inventory, the number of existing units and buildings, the status of upgrades and the number of units dedicated to attack the opponent (or to defend the base).

The dynamic system that governs the relationship between state and action variables in this paper is modeled with the following linear differential equations:

$$\begin{bmatrix} \mathbf{s}_{P1}(t_{k+1}) \\ \mathbf{s}_{P2}(t_{k+1}) \end{bmatrix} = \mathbf{A}\begin{bmatrix} \mathbf{s}_{P1}(t_k) \\ \mathbf{s}_{P2}(t_k) \end{bmatrix} + \mathbf{B}\begin{bmatrix} \mathbf{a}_{P1}(t_k) \\ \mathbf{a}_{P2}(t_k) \end{bmatrix} \tag{1}$$

where $\mathbf{a} = [\mathbf{a}_{P1}^T, \mathbf{a}_{P2}^T]^T$ and $\mathbf{s} = [\mathbf{s}_{P1}^T, \mathbf{s}_{P2}^T]^T$ are the vectors of time-dependent action and state variables from Player 1 (P_1) and Player 2 (P_2) at the strategic timescale, and $\mathbf{A}$ and $\mathbf{B}$ are the constant system matrices constructed based on the rules and mechanics of the game. These matrices define the relationships between states and actions. For instance, constructing a unit costs a certain amount of mineral, gas and supply. As another example, sending a worker to collect minerals increases the rate at which mineral is collected by a certain amount. These relationships are specified by the elements of these matrices. Also, the matrices $\mathbf{A}$ and $\mathbf{B}$ also represent the dependencies among the states and actions of both players. The description of the variables used to represent these states and actions are presented in Table 1. There are in total 86 state and 85 action variables for each player in this model. An

Table 1
State and action variables used to model strategic decisions in Starcraft II.

	Description	Qty
State	Quantity of available resources of each type	3
	Number of existing units and buildings of each type	34
	Status of upgrades of each type (binary)	30
	Number of units of each type in currently opponent territory	19
Action	Number of workers to send for harvesting resources of each type	2
	Number of new units and buildings of each type to create	34
	New upgrades of each type to purchase (binary)	30
	Number of new units of each type to send to opponent territory	19

extensive list of these variables and the all the values needed to construct matrices $\mathbf{A}$ and $\mathbf{B}$ are available in the public domain [25,35], and thus are omitted for brevity.

In this model, states can be considered as the representation of the current status of the system while actions represent the decisions to change this status. The first three states in Tbl. 1 corresponding to the resources in the game couple the decisions on most of the actions. Action variables can either directly or indirectly affect the game state. There is generally an action for each state that directly manipulates it. As an example, two types of resources, namely *mineral* and *gas*, (represented by two state variables) can be harvested with workers (represented by two action variables). On the other hand, the state of the resource *supply* increases by the action of constructing a building named supply depot. In this case, the action to directly manipulate the state for supply is part of the variable set that controls buildings.

This paper uses the publicly available gameplay data [35] to validate the dynamic system model and inform strategic decisions. This dataset stores the binary result of the game (win or loss) and all the detailed moves made by two opposing players throughout the game in the form of compact replay files. The state and action variables in Eq. (1) are extracted by post-processing these replay files using the publicly available API [35]. In the post-processing, time horizon is also converted to a gameplay progression between 0 and 1 by scaling the original time with the duration of each gameplay. The states and actions are recorded at fixed time intervals of 5% gameplay progression. The normalization on the time domain retains the sequentiality of the decisions and their relative time position in the overall problem horizon but loses the exact timing of the decisions to be made. Sections 2.3 and 3.1 discuss some implications of this normalization on the time horizon. In the post-processing, the state and action variables are normalized with respect to the maximum values in the dataset. Normalizing the input variables is a recommended step to improve the stability of many machine learning methods particularly when the scale of the input variables differs significantly [36]. For instance, in the present context, while the state variables for the status of upgrades are between 0 and 1, the state variable for the available supply ranges from 0 to 152 in the data.

A validation of this model with the actual results from a sample gameplay is shown in Fig. 2. The supply of a resource is increased by constructing particular buildings and consumed by training new units. Fig. 2 shows the status of this resource directly extracted from the game and compares it with the corresponding state value calculated from Eq. (1) using the actions tracked by the game API. The plot on the left shows the agreement between the mathematical model and the game results for a sample state in the game. Note that in the game, some actions have delayed responses that are not captured by the model in Eq. (1). For instance, constructing a supply depot costs mineral when construction starts while it contributes to supply when the construction completes. The original gameplay data has been shifted to remove time delays for this comparison. These time delays can significantly impact the operational decisions if the analysis is performed using a time step of a similar order of magnitude to these delays as shown in middle plot in Fig. 2. However, ignoring time delays does not constitute a major assumption for the analysis at the strategic timescale as shown by the plot on the right in Fig. 2 since the time steps are generally larger than these delays.

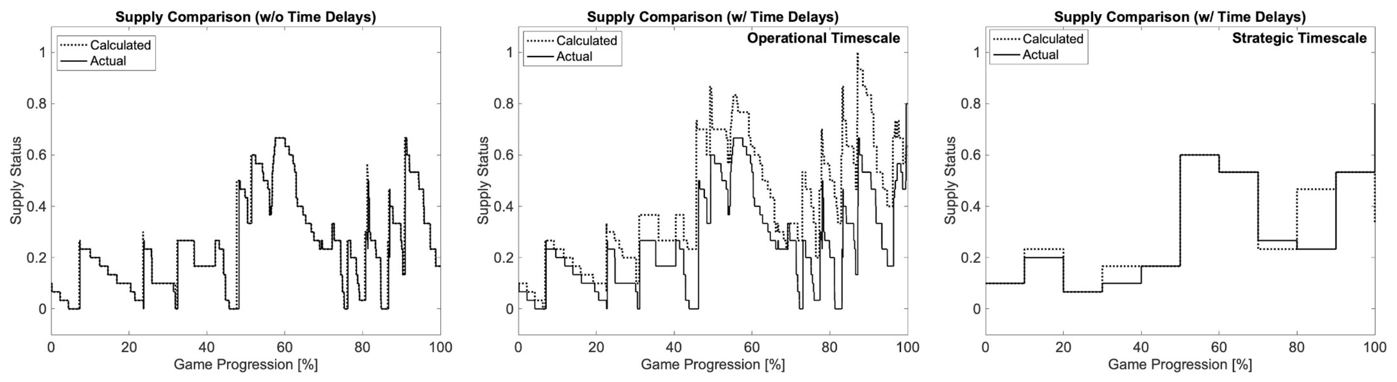


Fig. 2. Verification of the linear model in Eq. (1) by comparing a sample game state (i.e., supply status) calculated with Eq. (1) and the values extracted from the actual data of a sample game replay. Three figures represent comparison without time delays obtained by shifting the actual data (left), with time delays for an analysis with a time step of one second (middle), with time delays for an analysis with a time step of 10% game progression (right).

The time steps used in reward modeling in Section 2.3 and the problem solution in Section 3 are much larger compared to the ignored time delays.

2.2. Strategy clustering

Finding a single strategy that defeats all others rarely applies to real-world problems. Usually a multitude of strategies exists, and identifying these strategies is a key for success especially in competitive problems. This work clusters the gameplay data based on the strategies followed by previous game players and analyzes each strategy cluster separately to capture the variability of winning strategies. Simply, an optimal winning strategy for a cluster may be defeated by a strategy optimized for another cluster.

This application on Starcraft II uses the cumulative unit actions until 50% game progression to group the strategies. Unit actions, i.e., the decision variables corresponding to training units of each type, are integrated over time up to 50% game progression to represent the strategy of a player. A dyad of strategies from both players represent a data point used for clustering. A k-means clustering based on Euclidean distance [37] of vectors formed by this dyad is used. Note that the entire action variable set could have been used for representation, but the types of units trained can sufficiently identify different strategies in the particular case of Starcraft II. In general, the appropriate clustering method, distance measure and number of clusters vary [38] and should be chosen based on some expert guidance. In the Starcraft II data set for terran-vs-terran matches, five distinct strategy clusters can be identified as shown in Fig. 3. The horizontal axis represents different types of units in Starcraft II used for clustering and the vertical axis represents the quantities in proportion to the total number of units from both players. Commonly used units include workers such as *SCVs* and *MULEs* and lethal units such as *Marines* and *Siege Tanks*. Increasing the number of clusters beyond five creates clusters with similar strategies for this data set. In these plots, the opening strategies of winning players are on the right denoted by dashed lines and those of losing players are on the left denoted by solid lines. The unit quantities are reported in proportion to the total combined number of units from both players.

The clusters reveal a variety of strategies including offensive ones with a focus on *Marines* as in Strategy Cluster 2 and defensive ones with a focus on *SCVs* as in Strategy Cluster 1. Some clusters such as Cluster 1 identify gameplays where both players have a similar number of units. Note that the clustering results represent the state at mid-game, and the differentiation between winning and losing players may come in the second half of the gameplay.

2.3. Data-driven reward model

Finding the best set of actions to take at any point in time for DCPs requires an objective or *reward* function to assess the value of potential decisions. Many real-world problems do not have such an explicit objective but rather an end-goal such as defeating the opponent in case of Starcraft II. To construct a reward function, this paper uses the existing gameplay data mentioned in Section 2.1.

Assigning the value of 1 to a win and −1 to a loss, we introduce the following exponential function that back-propagates this end-game result to model the reward as a function of system states throughout gameplay progression.

$$f(\mathbf{s}(t_k)) = re^{\lambda(1-t_k)} \tag{2}$$

Here, r denotes the end-game result of 1 or −1; λ is a positive constant representing the backpropagation factor; t_k is the gameplay progression in $[0,1]$ that is normalized with respect to the duration of individual gameplays; $\mathbf{s}(t_k)$ denotes the normalized state vector including states from both sides; f is the reward function to train. The right-hand side in Eq. (2) takes a small value close to zero at the beginning of the gameplay ($t_k = 0$) and takes the value of r at the end ($t_k = 1$). Alternative approaches based on inverse reinforcement learning assume that the human or expert data is optimal with respect to the unknown reward function expressed as a function of given features [39]. This assumption is not applicable to the present problem context since the human data includes decisions from both experts and non-experts and some valuable information can be learned from non-experts such as the strategies that opponents use. While the present approach does not make an assumption regarding the expertise of the human participants, it uses an exponential, i.e., monotonically increasing or decreasing, reward function in Eq. (2) as a reference. This reference biases the trained reward to follow such a trend but the actual output of the training may not be monotonic with respect to time. The use of more advanced machine learning methods can be a subject for a future study.

A reward function for each strategy cluster should capture the time-dependencies among decisions in DCPs. Recurrent neural networks (RNN) have been developed in the machine learning literature to model sequential relationships in time-series data [40,41]. This study uses a long short-term memory (LSTM) network, which is a special type of RNN architecture [42]. LSTM networks are widely used for their ability to learn long-term dependencies in time series data, which classical applications of RNNs may fail to capture. These dependencies are crucial, especially for strategic analyses that span longer time periods.

A separate LSTM network is trained for each strategy cluster to estimate the term on the left-hand side of Eq. (2) as a function of the state variables. This approach assumes that the state variables used to model system dynamics at the strategic timescale could predict the game outcome, i.e., the reward. In this particular application, the effect of the operational decisions creates some notable variability in the outcome. Using the normalized game progression instead of actual time

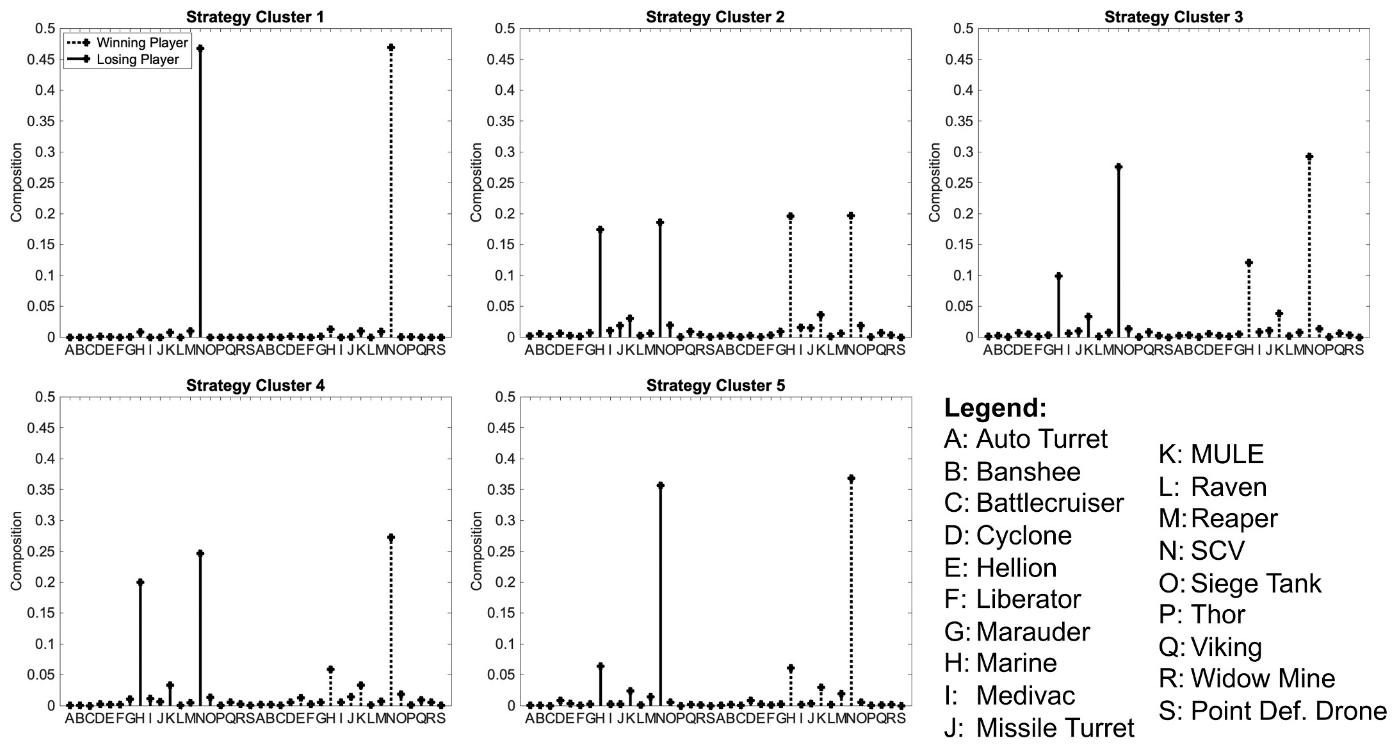


Fig. 3. Five strategy clusters identified with k-means clustering using cumulative actions for training units by both players until 50% game progression.

information simplifies the training of LSTM since all data points have the same number of time steps. Note that it does not restrict the generality of the present approach since LSTMs can also take a variable number of time steps as inputs. In some problems the actual time information can be critical while in this particular application, after a stable base is built, the sequence of decisions of the player and opponent matter more than the precise timing for long-term decision-making.

This study proposes several improvements to the training process. Unused state variables in each strategy cluster should be removed from the set of input variables before training the reward. For instance, the unit *Thor* has never been used by any player in Strategy Cluster 1 throughout the gameplay and it is removed from the reward model for that strategy cluster. Similar pre-processing can be applied to unpopular state variables if the prediction accuracy is low. For instance, the unit *Ghost* and the building *Ghost Academy* have been used by very few players and are not included in any of the strategy clusters.

The original data set contains only binary outputs of win or loss without any game resulting in a draw and it does not specify multiple levels of victory (e.g., winning slightly or winning by a large margin). Theoretically, if two players are in similar game states, these two players should have similar game scores. However, the model in Eq. (2) will represent that situation by assigning 1 to the winning and -1 to the losing player, resulting in a large difference in score for similar game states. In order to improve the model, a new set of synthetic data containing similar game states from both players throughout the gameplay (differed by a small random noise) is added to represent games ending in a tie. The synthetic data is created by taking the state vector of each player over the training time horizon and duplicating it for the opponent to create an equal state competition. A small random noise is added for one of the players to create some variation for training purposes. The process is repeated for each game play, i.e., the synthetic data increases the data size by two. The training treats synthetic data the same way as the regular data points where $r = 0$.

Finally, since the problem is a zero-sum game, the reward model should output the additive inverse of the original results when the definitions of Player 1 and 2 are flipped. To implement such a symmetry in the model, the original dataset can be mirrored by swapping the players in the state vector, i.e., $\mathbf{s}_1 = [\mathbf{s}_{P1}^T, \mathbf{s}_{P2}^T]^T$ and $\mathbf{s}_2 = [\mathbf{s}_{P2}^T, \mathbf{s}_{P1}^T]^T$, and by flipping the game scores, i.e., $r_1 = -r_2$. It can then be appended to the training data. Since this process only improves the symmetry with $f(\mathbf{s}_1(t_k)) \approx -f(\mathbf{s}_2(t_k))$ but does not guarantee it, we use the following average of two LSTM models queried with different player sequences as the reward function for perfect symmetry:

$$f(\mathbf{s}(t_k)) = f(\mathbf{s}_1(t_k)) - f(\mathbf{s}_2(t_k))/2 \tag{3}$$

Fig. 4 shows sample training results for Strategy Cluster 2 identified in Fig. 3. The solid line represents the exponential model assumed in Eq. (2), and the points in different tones of gray represent winning, losing and synthetic game data. The results show that the model can clearly distinguish the data in all three categories and that the continuous model output can be used to predict how much a player is expected to win. Predictions are not provided for less than 30% game progression because there is little differentiation between winning and losing players before that point. The LSTM network developed in this paper uses 100 hidden units, a dropout layer with a dropout probability of 0.5 (to prevent overfitting) and a fully connected layer at the end. The training uses an initial learning rate of 0.01, and the adams solver with a mean squared error loss function. These parameters as well as a proper batch size and the number of epochs for each strategy cluster have been tuned to improve the convergence and the accuracy with the test data. Also, adding more input variables to the training can potentially increase the prediction capability of the model at the expense of increasing problem complexity.

3. Control-based game theoretic decisions

This section presents a human-computer partnering framework to solve the problem modeled in Section 2 in two phases: In the initial non-adaptive phase, the human collaborator selects an opening strategy from a range of options extracted from the gameplay data and the computer then guides the human collaborator with suggestions to achieve some target goals defined by the selected opening strategy. In the subsequent adaptive phase with feedback control, the human collaborator is guided with game-theoretic suggestions based on the observed system state. The problem in this adaptive phase can be classified as a differential game which is a specific type of competitive problem applied to dynamical systems [10], allowing the integration of methods from both game and control theories. Details of both phases follow in the subsequent sections. The results presented in this section are entirely computational based on prior data collected from human players. A real-time experiment with a human in the loop is left to a future study.

3.1. Opening strategies

Initial decisions, referred to as opening strategies, in DCPs set the basis for a preliminary long-term strategy. These initial decisions may depend on the decision-maker's preference and evolve over time based on the opponent actions. Creating adaptive strategies based on the initial observations from the opponent may not be effective in this initial phase since these observations may not provide sufficient data to predict the opponent's long-term strategy. As an alternative, this study extracts standard opening strategies that led to successful outcomes in the empirical data. These strategies are then presented to a human at the beginning of the decision-making process as alternatives to choose from. It is necessary to semantically label these strategies for humans to make meaningful decisions among the options. After an opening strategy is selected, the computer guides the human in the hybrid team by suggesting target goals to achieve in terms of states defined in Eq. (1). Fig. 1 shows this process. In Starcraft II, a target goal could refer to building a certain number of units of a particular type. Once the present target goal is achieved, the computer prompts the human collaborator with the next set of goals until the final target goal in the opening strategy is achieved. Then, an adaptive phase based on game-theoretic solutions with feedback control follows this non-adaptive opening phase until the end.

Extraction of opening strategies is based on a sampling process used for strategy clustering described in Section 2.3. The cumulative decisions for each gameplay in a particular strategy cluster are sampled at evenly spaced gameplay progression intervals. The centroid of the sample set from each strategy cluster is presented as a set of suggestions in multiple stages in the opening phase. A sample opening strategy for Strategy Cluster 2 is shown in Fig. 5. Stage 0 in this figure represents the initial state of the game. The time interval to present the suggestions for the next stage varies depending on how quickly the human player achieves these targets. This opening uses only the unit actions, i.e., a subset of the action vector, for strategy targets. Using the full action vector can also be a viable alternative to guide the human player with more detailed suggestions. The in-situ performance of these alternative approaches should be determined in a human subject study considering the cognitive load they impose, an issue which is left to a future study.

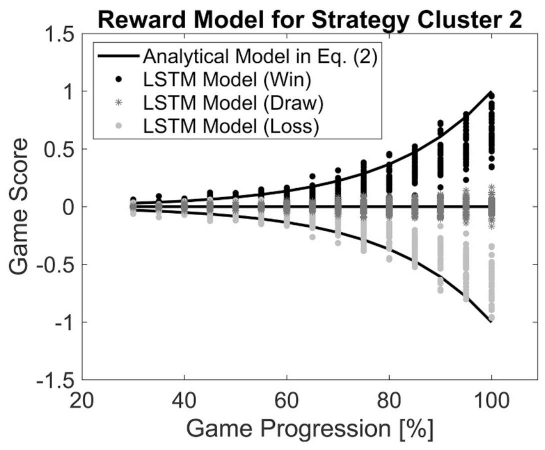


Fig. 4. Results of the LSTM network trained to model the reward for Strategy Cluster 2.

3.2. Adaptive strategies based on differential games

The adaptive phase that follows the successful completion of an opening strategy is based on a feedback control architecture that suggests game-theoretic decisions at the strategic timescale that counter the observed opponent state. Fig. 1 depicts this architecture. In a real-time application, the computer first identifies the strategy cluster closest to the observed game state and provides suggestions for that cluster by solving a differential game problem. Note that in this architecture, humans may or may not follow the suggestions from computers. Hence, the final decisions do not always have to be game-theoretic.

A differential game can be formulated for the DCP in this paper as follows:

$$\left.\begin{array}{ll}\max\limits_{\mathrm{a}_{p1}(t_k)} & \left\{\min\limits_{\mathrm{a}_{p2}(t_k)} \sum\limits_{j\in H} f\left(\mathrm{s}\left(t_j\right)\right)\right\}\\ \text{Subject to} & \mathrm{s}(t_{k+1}) = \mathrm{As}(t_k) + \mathrm{Ba}(t_k)\\ & -\epsilon_s \le \mathrm{s}(t_k) \le 1+\epsilon_s\\ & -\epsilon_{\mathrm{a}} \le a(t_k) \le 1+\epsilon_{\mathrm{a}}\\ & \mathrm{g}(\mathrm{s}(t_k),\mathrm{a}(t_k)) \le 0\\ & \mathrm{a}_i(t_k) \text{ is integer } \forall i\in I\\ & \left|f(\mathrm{s}(t_k))\right| \le \left(1+\epsilon_f\right)e^{\lambda(1-t_k)}\end{array}\right| \forall k \in H, \tag{4}$$

where $\mathscr{H}$ denotes the prediction horizon, t_k is the discrete time corresponding to the step k in the prediction horizon, ε_s, ε_a and ε_f are the tolerances to allow a small amount of violation in the bound constraints, and $\mathscr{I}$ is the set of action variables that can only take integer values (e.g., purchasing an upgrade). The first constraint is the dynamic system model, the second and third constraints are the bounds on state and action variables, the fourth constraint is the set of feasibility relationships that define possible actions for a given state, the fifth constraint is the integer variables, and the final constraint is the bound on the objective function to keep the search for optimal solutions within the valid region of the reward model. Since the problem is normalized with respect to the maximum values that appear in the dataset, tolerances might be used on the bounds to allow the problem search to go slightly beyond these values. The number of design variables, i.e., $\mathbf{a}_{P1}(t_k)$ and $\mathbf{a}_{P2}(t_k)$, varies among different strategy clusters. Note that the nested optimization in Eq. (4) implies that the actions of one player have to depend on the actions the other player.

Depending on the nature of the objective function and the dynamic

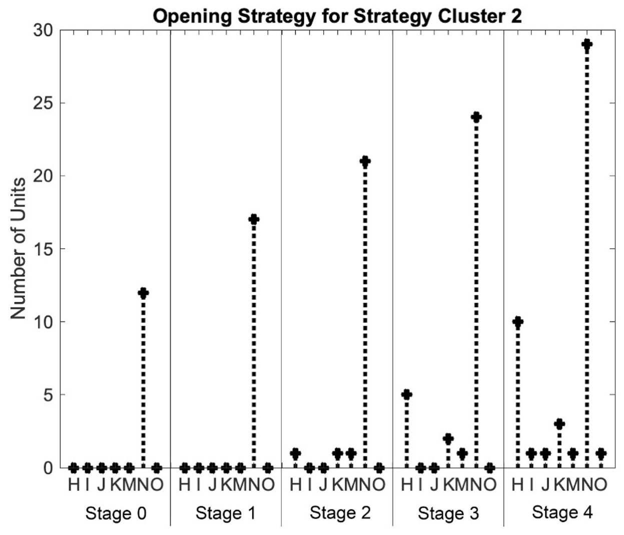


Fig. 5. Target goals defined by the opening strategy for Strategy Cluster 2 (Legend for the horizontal axis is provided in Fig. 3.

system model an appropriate control method can be employed to solve the optimization problem for both sides. For instance, linear quadratic regulators are commonly used for problems with quadratic objective functions and linear system models [16]. Also, dynamic programing can be used for more general applications with a relatively small number of state and action variables [17]. This paper uses a model predictive control [11] strategy to solve a generalized nonlinear control problem over a finite time period by utilizing global optimization methods developed for mixed-integer problems. Model predictive control solves the problem in Eq. (4) over a fixed small prediction horizon, applies the resulting control actions to the current time step to move the system to the next state and shifts the prediction time horizon forward to repeat the process.

Finding a Nash equilibrium in complex DCPs such as Starcraft II requires an iterative solution of the problem in Eq. (4) for both sides until convergence. This approach is also known as a relaxation method [43]. In that approach, the opponent actions in the inner optimization problem are kept constant when searching for the optimal control actions for one side, and the resulting solutions are used as fixed opponent actions when the same problem is solved for the other side. The process repeats until no further improvement is observed in the objective function for both sides. Considering the number of iterations required for the convergence and high dimensionality of complex problems, the prediction horizon in the model predictive control problem must be limited to obtain a solution within reasonable computation time.

This section applies the model predictive control-based all-in-one approach to the gameplay data from Starcraft II to iteratively search for game-theoretic actions at each time step. Note that the data set belongs to past gameplays that already define the system state over the entire time horizon. In this work, an independent problem (over a fixed prediction horizon) is solved for each time step starting with the existing game state from the data instead of evolving the game state with resulting optimal decisions. This approach allows a fair comparison between the decisions made by actual players and the game-theoretic decisions throughout the gameplay.

Fig. 6 shows the game-theoretic decisions for the gameplay data in Strategy Cluster 2 compared to the actual player decisions. Control problem at each time step is solved with a genetic algorithm to obtain these results. In order to improve the convergence of the iterative solution of the Nash equilibrium problem, the genetic algorithm is seeded with the best 50% of the solutions in the final population from the previous iteration. Seeding genetic algorithms with known good solutions is commonly used in the literature to improve convergence [44]. The prediction horizon is set to the next three time-steps where a time-step is 10% gameplay progression. This prediction horizon marks 80% gameplay progression as the final time-step for problem solution since, after that time point, model predictive control requires predicting the game state and reward beyond 100% gameplay progression. The algorithm uses a population size of 1000 and 250 generations. In this problem, five iterations are sufficient to find an equilibrium solution between two players. All these parameters should be adjusted in other applications to improve the convergence of the solution algorithm.

Fig. 6 shows that the equilibrium solution favors a more “balanced” game reward on average compared to the actual reward achieved by the players. At each time step, the game-theoretic decisions yield a higher reward for the losing players and lower reward for the winning ones in return. It is an expected outcome since the game-theoretic solution optimizes the decisions for both players. For instance, the lower game reward on average for winning players should be interpreted as the best action to take assuming their opponents make the best decisions for themselves. In reality, the players do not always make rational decisions, which leads to the difference between the equilibrium and actual player rewards. Considering the fact that the players who lose the game make bad decisions, the game-theoretic solutions might be overly conservative for winning players. For engineering applications under market competition or military applications under adversarial combat, the present approach is most beneficial for inexperienced decision-makers who need reasonable guidance. The approach is also valuable for expert decision-makers when the information regarding the competitor/opponent strategy is not readily available. For such cases, the game theoretic approach provides safe (conservative), though suboptimal, suggestions, by assuming the competitor/opponent strategy is close to optimal with respect to the objective function learned from the prior data.

The present approach to making strategic decisions in DCPs is computationally expensive. Thus, the problem in Eq. (4) must be solved offline to find game-theoretic actions for a range of potential game states in each strategy cluster. A real-time implementation for human-computer collaboration is possible by fitting a response surface model to these solutions and querying it online. Since a real-time human-in-the-loop implementation requires incorporating human factors into decision-making, it is left as a topic for future work.

4. Partitioned problem-solving

The differential game problem formulated in Eq. (4) may not be effectively or efficiently solved with an all-in-one approach that assumes a single decision-maker. Recall that the original problem is separated into strategic and operational decisions to enable a simpler problem formulation that can use different solution strategies for these two subproblems. Even after this separation, the strategic decisions define a large problem space considering the number of decisions to make over multiple time steps. Searching solutions efficiently in that large space is challenging and may not be possible for real-world problems for which real-time solutions are desired. This section examines the benefits of further partitioning the strategic decisions into smaller and more manageable subproblems to be solved by different computer agents in the hybrid team. This partitioning approach represents a systematic method to perform the division of labor among a team of computer agents in a hybrid team. While the methods presented in this paper can be applied to divide labor among human members in the team, the results are expected to be quite different for humans and computers. Analysis of task partitioning for humans remains for future work.

Here, task partitioning is defined as separating the overall set of action variables into a finite number of variable groups, each of which represents a subproblem to be solved by a computer agent. A collection of variable groups is referred to as a partition. Task partitioning should be applied to each strategy cluster identified in Section 2.3 separately since the interactions among variables differ significantly between

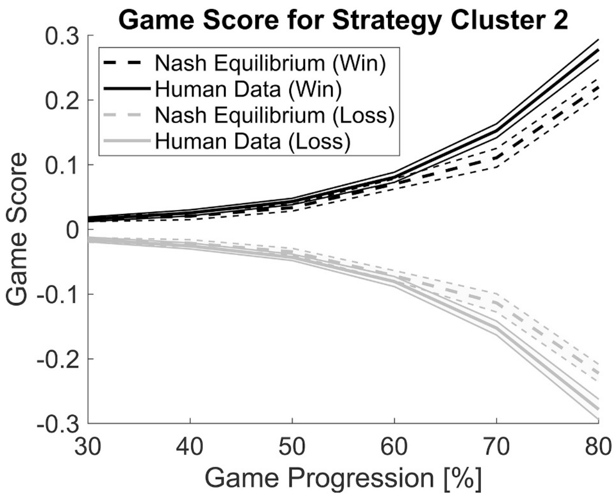


Fig. 6. Comparison of Nash equilibrium solution with human player data for Strategy Cluster 2. Bold lines and shaded areas correspond to mean and standard error, respectively.

strategy clusters. When performing this task partitioning, a systematic method is necessary to find the grouping that will maximize the team performance. The size of the partitions and the couplings among the decisions are known to have an impact on the overall performance [45,46]. Existing approaches based on design structure matrices [45] or functional dependency tables [46] to problem partitioning are not applicable to problems without explicitly defined objectives. Further, task allocation methods for networked computing systems only apply to well-defined tasks with known dependencies [47]. The data-driven objectives presented in Section 2.3 are black-box functions without an explicit expression or known dependencies to the decision variables.

This section presents three distinct methods for task partitioning. The *manual* partitioning is a semantically meaningful partitioning of the problem and is used as a baseline approach for comparison. The *unsupervised* partitioning is a data-driven approach based on the variable correlations in the empirical data. The *supervised* partitioning is an optimization-based method that uses the data-driven objective to align individual objectives of the team members with the overall team goal. These three methods represent a spectrum from a completely ad hoc to a completely objective way of performing task partitioning.

Using these partitions to make decisions as a team requires a coordination method. Since the decisions in each subproblem (variable group) are coupled, a coordination strategy must interface the solutions obtained from each subproblem and ensure consistency. Studies in the decomposition-based design optimization literature have introduced several coordination methods including single-level formulations that coordinate the solutions in a single optimization problem [48] and multi-level formulations that solve different optimization problems for each variable cluster [49]. This study employs a simple sequential coordination strategy that solves the optimization problem for each subproblem in sequence by using the optimal solutions from the preceding subproblems. This strategy is useful to reduce the computation time of the overall solution. While such a simple strategy is not guaranteed to always find optimal solutions for the integrated problem, the results presented in this section shows the effectiveness of this approach over an all-in-one approach.

4.1. Manual partitioning

A semantic partitioning based on the meanings of the variables is an ad hoc approach that can serve as a baseline scenario. There are many partitioning solutions possible due to the subjectivity of this approach. Such an approach might be preferable for human members in hybrid teams for its intuitive nature. Here, it provides a useful reference to compare with the other two methods presented in this section that are developed specifically for computer teams. Table 2 shows the descriptions of four subproblems defined by variable groups manually created based on the semantic meaning of the action variables in Starcraft II. Fig. 7 also depicts the distribution of action variables across four subproblems shown by different tones of gray, demonstrating that the three approaches provide distinct patterns. For a fair comparison with the other two approaches, manual partitioning is performed to balance the number of variables in each subproblem without any overlapping variables between subproblems.

4.2. Unsupervised partitioning

An automated partitioning approach that uses prior experimental data can reduce the subjectivity of the partitioning decisions. Both k-means and k-medoids clustering have been used in the literature to group data into a fixed number of sets [38]. The advantage of the k-medoids algorithm over k-means is the ability to use custom distance measures other than Euclidean distance. The correlations among the action variables in the empirical data can provide useful information regarding how the decisions are made by the participants. Here, the Pearson correlation is proposed for use in a k-medoids clustering algorithm [50] to group the variables for subproblems. This distance measure favors highly correlated variables in the dataset to appear in the same subproblem.

This work partitions the action variable set into four groups with k-medoids. This method is referred to as unsupervised since it is not informed by the reward function and instead is based only on correlations between action variables. Hence, the effectiveness of the unsupervised partitioning decisions highly depends on the dataset. The partitioning results presented in Fig. 7 show that semantically partitioned variables are dispersed in multiple subproblems.

4.3. Supervised partitioning

A robust partitioning strategy should use the reward function to identify the relationships among the variables. Finding the task partitioning that maximizes the team performance by solving the Nash equilibrium problem in an inner loop is computationally expensive. This work proposes a computationally efficient proxy to represent the effectiveness of the team decisions without solving the Nash equilibrium problem for multiple gameplays.

The proposed method trains separate objective functions for individuals in the team using only the state variables assigned to that individual. The training process is the same as in Section 2.3 with the only difference being the variables used for training. The training errors from each variable cluster correspond to the inability of each cluster to predict the team objective, i.e., a misalignment. The total sum of misalignments between the objectives of the individuals and the team is used as an objective to make partitioning decisions in an optimization problem. Notice that this partitioning is based on the state variables rather than action variables. Since each state variable can be directly manipulated by an action variable, the partitioning based on state variables can be mapped to action variables using that relationship. In a general case where some of the states or actions are strongly linked to multiple actions or states, overlapping subproblems that share action variables might be necessary. For simplicity, this study does not allow subproblems with overlapping variables. The optimization problem for task partitioning can be formulated as follows:

$$\begin{aligned}
&\underset{\mathbf{p}}{min} && \sum_{n=1}^{N}\left(f_n(\mathbf{I}_n(\mathbf{p})\cdot\mathbf{s}(t)\,) - re^{\lambda(1-t)}\,\right)^2 \\
&\text{where} && \mathbf{I}_n(\mathbf{p}) = \begin{bmatrix} \cdots \\ \mathbf{v}_i \\ \cdots \end{bmatrix} \forall i \in j | p_j = n \\
&\text{subject to} && \mathbf{1} \leq \mathbf{p} \leq \mathbf{1}N \\
& && \mathbf{p}\ \text{is integer}
\end{aligned} \tag{5}$$

where **p** represents the vector mapping each state variable to a subproblem, N is the given number of subproblems, f_n is the individual objective function trained for the subproblem n, v_i is the i^{th} row of the identity matrix, and $\mathbf{I}_n$ is the binary indicator matrix that maps variables to subproblems. Here, $\mathbf{I}_n(\mathbf{p})\cdot\mathbf{s}$ gives the vector of state variables belonging to the subproblem n.

The problem in Eq. (5) can be solved with a particle swarm optimization method by relaxing the integer constraints and rounding the continuous design variables to the nearest integer. It has been shown that particle swarm optimization with continuous relaxation performs as effectively as a branch and bound method for integer programming

Table 2
Description of the variable groups created by the manual partitioning.

Group	Description	Qty
1	Action variables for harvesting resources and constructing buildings	17
2	Action variables for training units	19
3	Action variables for upgrade purchases	30
4	Action variables for relocating units	19

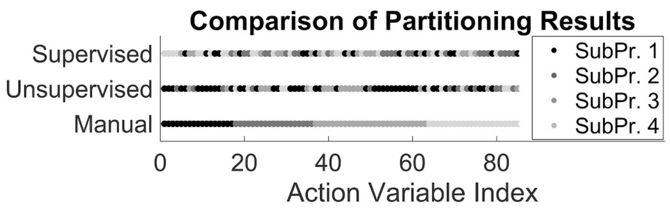


Fig. 7. Variable distributions across subproblems in three different methods. Different tones of gray represent the mapping from action variables to four subproblems.

problems [51]. Fig. 8a shows the optimization results in comparison to the manual and unsupervised partitioning results. Supervised partitioning improves the total misalignment significantly compared to other partitioning methods.

The performance of the three partitioning methods when making decisions for the actual DCP is depicted in Fig. 8b. The optimal rewards are compared for winning players only. These results are obtained by keeping the opponent decisions constant at the Nash equilibrium solution obtained with the non-partitioned approach. By definition of the Nash equilibrium, the all-in-one approach cannot improve the results further without changing the decisions of the opponent while partitioned approaches are able to find better solutions for the winning players as seen in Fig. 8b. It is likely that the all-in-one solution is one of the many local minima in the large solution space. The results in Fig. 8a and b also indicate that from manual to unsupervised to supervised partitioning the team performance increases as the total misalignment introduced in this section decreases, albeit showing a nonlinear relationship between the two.

5. Discussion

This section summarizes the important outcomes and key take-aways from this research. The present methodology is applicable to a wide range of engineering problems where historical data from human decision-makers either exists or can be collected from an actual system or from a virtual environment through experiments. As in most of the machine learning approaches, the quality of the solution depends on the input data. The input dataset should include a range of different strategies (including good and bad ones) in order for the computer system to learn how to properly adapt to the opponent strategy. The size of the amount of data needed increases based on the size of the state space. Also, the data should include the important features that can predict the quality of the solution. The following are the major take-aways from the present results.

Separation of strategic and operational decisions enables the effective management of problem complexity

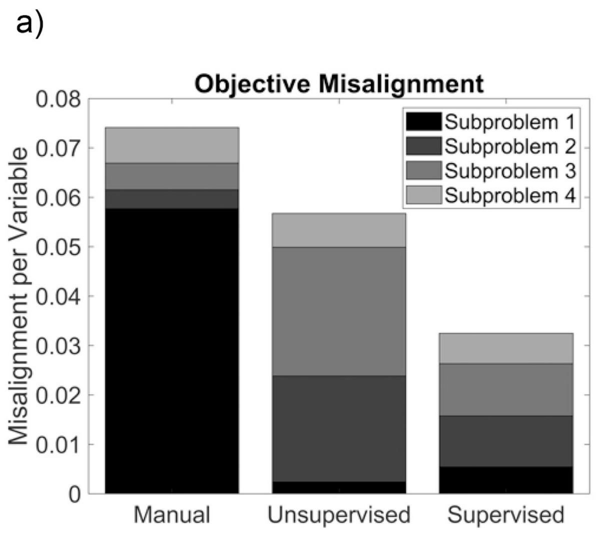
This distinction allows separate analyses for these two fundamentally different problems. The present study shows that a strategic decision-making problem can be isolated from operational decisions, and a computer informed by prior data can learn objectives and find solutions for this problem in a computationally efficient way. Without this separation, every new unit in the army would add a new variable to the problem while, in the present approach, the problem size for strategic decisions remains constant regardless of the army size. The strategic decisions determine the problem domain for the operational decisions. For instance, given the strategic decision to train a number of units of a particular type in Starcraft II, the operational problem to control these fixed number of units can be formulated as a well-defined optimization problem or be left to the human player if the size of the state space is too large for the computer.

The methodology to develop computer processes for strategic decisions is able to adapt to the changes in the opponent strategy

The modeling process used in this research captures the possibility of multiple winning strategies for DCPs. Two adaptation mechanisms allow computers to find the most effective decisions that counter the opponent strategy. The first mechanism chooses the proper strategy cluster based on the system state to search for solutions. The present method identifies five strategy clusters to be selected from before solving the differential game problem. A radical change in the opponent strategy can move the system state to another cluster resulting in major changes in computer decisions. The second mechanism finds the game-theoretic solutions within a cluster using a state feedback controller. The solution of Eq. (4) depends on the real-time decisions of the opponent. The adaptations are minor in this mechanism compared to the first one.

The human-computer collaboration architecture can mitigate some of the limitations of the game theory

The suggestions provided by the computer assume rational decision-making by the opponent with respect to the reward function trained from the experimental data. On one hand, this assumption leads to conservative solutions in Fig. 6 for the worst-case scenarios and does not account for the expertise level of the opponent. On the other hand, the present architecture in Fig. 1d–f leaves the final decisions to a human



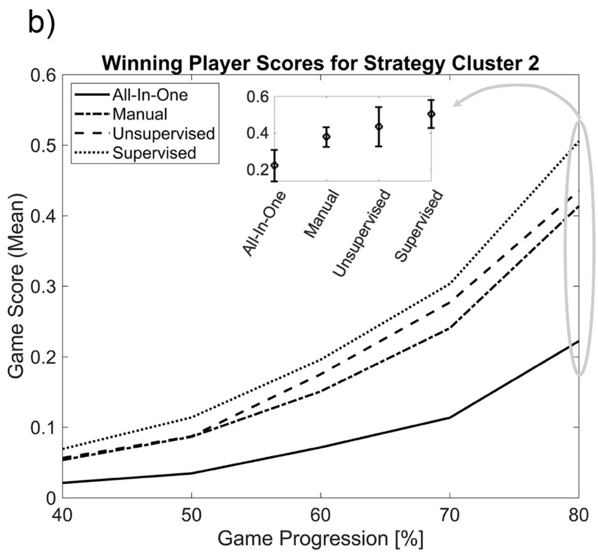


Fig. 8. (a) Comparison of total misalignment in different partitioning decisions. (b) Optimal reward results for winning players in Strategy Cluster 2 obtained from different partitioning methods.

who may or may not follow the game-theoretic suggestions. The human can interpret the opponent behavior and make an informed final decision considering the suggestions from the AI without necessarily following them all the time.

The complexity of DCPs favors a team of computers over a single computer to make effective strategic decisions

The results in Fig. 8b show a significant improvement in performance with partitioned problem solutions compared to an all-in-one approach. In problems with large decision spaces, an all-in-one approach may find a local suboptimal solution. Problem partitioning allows a more efficient search in this space. Therefore, this research suggests that future hybrid teams should utilize an advisory board of computers for large-scale problems as opposed to a single meta-planner computer. This research also suggests two alternatives to distribute the task among these computers. While supervised partitioning provides slightly better performance than unsupervised partitioning, the computational cost is also higher. For a general problem, it is suggested to start with unsupervised partitioning. If the performance is not satisfactory, the result can be used as a starting point for supervised partitioning.

Computer team performance improves with non-intuitive task partitioning among multiple computers

The team performance results in Fig. 8b show that both unsupervised and supervised partitioning outperform a manual semantically meaningful partitioning approach. Also, Fig. 7 shows that there is no clear pattern in unsupervised and supervised partitioning. These findings suggest that computer teams favor unconventional task distributions for effective decision-making in hybrid teams. These results indicate that humans and computers might have different preferences regarding how the tasks should be allocated between members of a hybrid team. Finding the optimal partitioning and division of labor for synergistic collaboration between humans and computers remains an open research question.

6. Conclusion

This paper presents an architecture for human-computer collaboration and a methodology to develop computer processes to make strategic design decisions for large-scale DCPs. The presented approach integrates sequence learning, model predictive control, and game-theory to make adaptive decisions. The paper also discusses the value of computer teams to making effective design decisions under complex problems and presents multiple approaches for problem partitioning to assign separate subproblems to the individual computers in these teams. The application of the generalized methodology has been illustrated on the video game Starcraft II as a representative of a complex problem-solving situations under dynamic competition.

The results show that the game-theoretic solutions yield a higher reward on average for the losing players and lower reward for the winning players, suggesting that game theory can be overly conservative for high-performing players. The present approach can provide useful guidance for non-expert decision-makers and provide safe (conservative), though suboptimal suggestions for experts particularly when competing against an opponent with an unknown level of expertise. The game-theoretic results further suggest that appropriateness of the game-theoretic solutions should be assessed by a human for improved performance. The partitioned decision-making results show that partitioned problem solutions with multiple computer agents outperform an all-in-one solution with a single agent. While coordination of partitioned problems could hinder the performance, the results suggest that the benefits of partitioning surpass the drawbacks particularly in large-scale complex problems.

Declarations of interest

None.

Acknowledgment

This work was supported in part by the Defense Advanced Research Projects Agency under Cooperative Agreement N66001-17-2-4064.

References

[1] P.M. Fitts, Human engineering for an effective air navigation and traffic control system, Tech. rep., National Research Council Committee on Aviation Psychology, Washington, DC, 1951.

[2] J.C. de Winter, D. Dodou, Why the fitts list has persisted throughout the history of function allocation, Cogn. Tech. Work 16 (1) (2014) 1–11.

[3] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, D. Hassabis, A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play, Science 362 (6419) (2018) 1140–1144.

[4] N. Ain, G. Vaia, W.H. DeLone, M. Waheed, Two decades of research on business intelligence system adoption, utilization and success–a systematic literature review, Decis. Support. Syst. 125 (2019) 113113.

[5] R. Parasuraman, V. Riley, Humans and automation: Use, misuse, disuse, abuse, Hum. Factors 39 (2) (1997) 230–253.

[6] J.Y.C. Chen, M.J. Barnes, Human–agent teaming for multirobot control: a review of human factors issues, IEEE Trans. Human-Machine Syst. 44 (1) (2014) 13–29.

[7] M. Wei, G. Chen, J.B. Cruz Jr., L. Hayes, M. Kruger, E. Blasch, Game-theoretic modeling and control of military operations with partially emotional civilian players, Decis. Support. Syst. 44 (3) (2008) 565–579.

[8] K.W. Hipel, M.M. Jamshidi, J.M. Tien, C.C. White III, The future of systems, man, and cybernetics: application domains and research methods, IEEE Trans. Syst. Man Cybern. Part C Appl. Rev. 37 (5) (2007) 726–743.

[9] G. Ghiani, G. Laporte, R. Musmanno, Introduction to Logistics Systems Planning and Control, John Wiley & Sons, West Sussex, England, 2004.

[10] R. Isaac, Differential Games: A Mathematical Theory with Applications to Warfare and Pursuit, Control and Optimization, John Wiley & Sons, New York, NY, 1965.

[11] C.E. Garcia, D.M. Prett, M. Morari, Model predictive control: Theory and practice—a survey, Automatica 25 (3) (1989) 335–348.

[12] A.E. Bayrak, C. McComb, J. Cagan, K. Kenneth, A differential game approach to dynamic competitive decisions toward human-computer collaboration, in: ASME International Design Engineering Technical Conference, Anaheim, CA, 2019, Aug 18–21, 2019 p. V007T06A017, Anaheim, CA.

[13] P.A. Ioannou, S. Jing, Robust Adaptive Control, Prentice Hall, Upper Saddle River, NJ, 1996.

[14] R.B. Myerson, Game Theory: Analysis of Conflict, Harvard University Press, Cambridge, MA, 1997.

[15] E.J. Dockner, S. Jørgensen, N. Van Long, G. Sorger, Differential Games in Economics and Management Science, Cambridge University Press, Cambridge, UK, 2000.

[16] P. Bernhard, Linear-quadratic, two-person, zero-sum differential games: necessary and sufficient conditions, J. Optim. Theory Appl. 27 (1) (1979) 51–69.

[17] H. Zhang, Q. Wei, D. Liu, An iterative adaptive dynamic programming method for solving a class of nonlinear zero-sum differential games, Automatica 47 (1) (2011) 207–214.

[18] C. McComb, J. Cagan, K. Kotovsky, Lifting the veil: drawing insights about design teams from a cognitively-inspired computational model, Des. Stud. 40 (2015) 119–142.

[19] N.J. McNeese, M. Demir, N.J. Cooke, C. Myers, Teaming with a synthetic teammate: insights into human-autonomy teaming, Hum. Factors 60 (2) (2018) 262–273.

[20] F. Gao, M.L. Cummings, E. Solovey, Designing for robust and effective teamwork in human-agent teams, in: Robust Intelligence and Trust in Autonomous Systems, Springer, Boston, MA, 2016, pp. 167–190.

[21] T.B. Sheridan, Supervisory Control, in: Handbook of Human Factors and Ergonomics, John Wiley & Sons, Inc., Hoboken, NJ, 2006, pp. 1025–1052.

[22] P.A. Hancock, D.R. Billings, K.E. Schaefer, J.Y.C. Chen, E.J. de Visser, R. Parasuraman, A meta-analysis of factors affecting trust in human-robot interaction, Human Fact. J. Human Fact. Ergonom. Soc. 53 (5) (2011) 517–527.

[23] L. Goutas, B. Hess, J. Sutanto, If erring is human, is system use divine? Omission errors during post-adoptive system use, Decis. Support. Syst. 130 (2020) 113225.

[24] J.E. Mercado, M.A. Rupp, J.Y. Chen, M.J. Barnes, D. Barber, K. Procci, Intelligent agent transparency in human-agent teaming for multi-UxV management, Hum. Factors 58 (3) (2015) 401–415.

[25] Blizzard Entertainment Inc, Starcraft II, available at: https://starcraft2.com/en-us/, 2018 (accessed in Jul 2018).

[26] S.E. Page, Diversity and Complexity 2, Princeton University Press, Princeton, NJ, 2010.

[27] J. Ladyman, J. Lambert, K. Wiesner, What is a complex system? Eur. J. Philos. Sci. 3 (1) (2013) 33–67.

[28] C.B. Browne, E. Powley, D. Whitehouse, S.M. Lucas, P.I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, S. Colton, A survey of Monte Carlo tree search methods, IEEE Trans. Comput. Intell. AI Games 4 (1) (2012) 1–43.

[29] S. Gelly, L. Kocsis, M. Schoenauer, M. Sebag, D. Silver, C. Szepesvári, O. Teytaud, The grand challenge of computer go: Monte Carlo tree search and extensions, Commun. ACM 55 (3) (2012) 106.

[30] O. Vinyals, T. Ewalds, S. Bartunov, P. Georgiev, A.S. Vezhnevets, M. Yeo, A. Makhzani, H. Küttler, J. Agapiou, J. Schrittwieser, et al., Starcraft II: A new

challenge for reinforcement learning, Tech. rep., Deep Mind & Blizzard, available at: http://arxiv.org/abs/1708.04782, 2017.

[31] D. Churchill, M. Preuss, F. Richoux, G. Synnaeve, A. Uriarte, S. Ontañnón, M. Čertický, Starcraft bots ad competitions, in: Encyclopedia of Computer Graphics and Games, Springer, Cham, 2016, pp. 1–18.

[32] O. Vinyals, I. Babuschkin, W.M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, et al., Grandmaster level in starcraft ii using multi-agent reinforcement learning, Nature 575 (7782) (2019) 350–354.

[33] S. Verboven, J. Berrevoets, C. Wuytens, B. Baesens, W. Verbeke, Autoencoders for strategic decision support, Decis. Support. Syst. (2020), https://doi.org/10.1016/j.dss.2020.113422.

[34] P.T. Grogan, A.E. Bayrak, Operational and strategic decisions in engineering design games, in: ASME International Design Engineering Technical Conference, Quebec City, Canada, 2018, Aug 26–29, 2018 p. V02BT03A024, Quebec City, CN.

[35] Blizzard Entertainment Inc, Starcraft II client - C++ library supported on windows, linux and mac designed for building scripted bots and research using the sc2api, available at: https://github.com/Blizzard/s2client-api, 2018 (accessed in Jul 2018).

[36] C.M. Bishop, Neural Networks for Pattern Recognition, Oxford university press, Cambridge, UK, 1995.

[37] S. Lloyd, Least squares quantization in pcm, IEEE Trans. Inf. Theory 28 (2) (1982) 129–137.

[38] T. Warren Liao, Clustering of time series data—a survey, Pattern Recogn. 38 (11) (2005) 1857–1874.

[39] A.Y. Ng, S.J. Russell, Algorithms for Inverse Reinforcement Learning, in: Seventeenth International Conference on Machine Learning, Morgan Kaufmann Publishers Inc, San Francisco, CA, 2000, pp. 663–670.

[40] J.J. Hopfield, Neural networks and physical systems with emergent collective computational abilities, Proc. Natl. Acad. Sci. 79 (8) (1982) 2554–2558.

[41] D.E. Rumelhart, G.E. Hinton, R.J. Williams, Learning representations by back-propagating errors, Nature 323 (6088) (1986) 533–536.

[42] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural Comput. 9 (8) (1997) 1735–1780.

[43] P.T. Harker, A variational inequality approach for the determination of oligopolistic market equilibrium, Math. Program. 30 (1) (1984) 105–111.

[44] R.L. Haupt, S.E. Haupt, Practical Genetic Algorithms, John Wiley & Sons, Inc., Hoboken, NJ, 2003.

[45] S.D. Eppinger, T.R. Browning, Design Structure Matrix Methods and Applications, The MIT Press, Cambridge, MA, 2012.

[46] R.S. Krishnamachari, P.Y. Papalambros, Hierarchical decomposition synthesis in optimal systems design, J. Mech. Des. 119 (4) (1997) 448.

[47] M. Grum, B. Bender, A.S. Alfa, N. Gronau, A decision maxim for efficient task realization within analytical network infrastructures, Decis. Support. Syst. 112 (2018) 48–59.

[48] J.R.R.A. Martins, A.B. Lambe, Multidisciplinary design optimization: a survey of architectures, AIAA J. 51 (9) (2013) 2049–2075.

[49] J.T. Allison, M. Kokkolaras, P.Y. Papalambros, Optimal partitioning and coordination decisions in decomposition-based design optimization, J. Mech. Des. 131 (8) (2009), 081008.

[50] H.-S. Park, C.-H. Jun, A simple and fast algorithm for k-medoids clustering, Expert Syst. Appl. 36 (2) (2009) 3336–3341.

[51] K.E. Parsopoulos, M. Vrahatis, Recent approaches to global optimization problems through particle swarm optimization, Nat. Comput. 1 (2–3) (2002) 235–306.