IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2023.0322000

Resilience for Goal-Based Agents: Formalism, Metrics, and Case Studies

JENNIFER LEAF[1], (Member, IEEE), JULIE A. ADAMS[1], (Senior Member, IEEE), MATTHIAS SCHEUTZ[2], (Senior Member, IEEE) and MICHAEL A. GOODRICH[3], (Senior Member, IEEE)

[1]Collaboative Robotics and Intelligent Systems Institute, Oregon State Univeristy, Corvallis, OR 97331 USA (e-mail: leafj@oregonstate.edu, adamjuli@oregonstate.edu)
[2]Department of Computer Science, Tufts University, Medford, MA 02155 USA (e-mail: Matthias.Scheutz@tufts.edu)
[3]Department of Computer Science, Brigham Young University, Provo, UT 84602 USA (e-mail: Mike@cs.byu.edu)

Corresponding author: Jennifer Leaf (e-mail: leafj@oregonstate.edu).

This work was supported in part by the Office of Naval Research grant #N00014-18-1-2831. Leaf also received a Foundation Scholar award from the Achievement Rewards for College Scientists.

ABSTRACT Goal-based agents need to be resilient to perturbations in the world. Existing resilience definitions emphasize *maintenance-type goals* and, consequently, describe how well systems can recover and return to a desirable operating state after a perturbation. An alternative formulation of resilience is required for *achievement-type goals* that emphasize the ability to progress towards a goal state. This manuscript proposes a new formalism of resilience as a computational construct that accounts for an agent's sensors, effectors, communication channels, and computational resources. Two metrics for comparing the resilience of different algorithms are derived, namely *power* and *efficiency*. Three case studies demonstrate how the metrics can be used to characterize power-efficiency tradeoffs in algorithm design. A common property of the resilient algorithms in the case studies is that they have the ability to exploit many possible world trajectories, often at the cost of failing to find optimal trajectories in unperturbed conditions.

INDEX TERMS Resilience, Perturbations, Single Goal, Agents, Robots.

I. INTRODUCTION

AGENT-based systems must be able to achieve their goals even when operating conditions cannot be predicted completely. The pursuit of optimality, with the objective of implementing the best algorithm given the agent's computational system and task parameters, imposes constraints that can reduce the agent's ability to cope with unexpected events. Indeed, real-world robot deployments have demonstrated that cutting-edge algorithms, extensive simulation, and testing alone do not guarantee successful mission performance because unanticipated or unmodeled perturbations frequently occur [1], [2]. A *resilient* agent achieves its goals under unanticipated perturbations, even if redundant or suboptimal behaviors are necessary to achieve those goals.

Resilience as a concept has been applied across many domains, including engineering, ecology, psychology, sociology, and economics (e.g., [3]–[12]). Two types of resilience definitions are particularly relevant, each of which is based on the notion of an equilibrium. First, *engineering resilience* quantifies how well systems are able to recover performance after a perturbation and return to an equilibrium [3]. Engineering resilience assumes the equilibrium is a desirable state and leads to algorithm designs that seek to maintain that equilibrium. Second, *ecological resilience* describes whether relationships between system elements can persist over time [13], [14], such that system elements remain in a consistent state space region. Ecological resilience assumes that the relationship between system elements is configured in a desirable way and seeks to maintain that configuration.

Engineering and ecological resilience are suited for systems that pursue *maintenance* or *procedural* goals [15], in which the system either needs to maintain a desired state or perform a specified set of actions, respectively. Resilience is not as clearly defined for agents that pursue *achievement*-oriented goals in which the goal is achieved if the system reaches a state in which a task has been accomplished.

Consider an autonomous vehicle that after delivering goods needs to traverse mountainous terrain to return to its home base. While following the road on the vehicle's map, it encounters a boulder blocking the road at the entrance to a narrow canyon, preventing it from continuing. The vehicle must handle this environmental "perturbation" (e.g., determine alternative off-road paths to potentially traverse). Driving up a steep incline requires the vehicle to lower its load (i.e., drop-

ping all empty containers) in order to make it up the mountain. After a few miles, the vehicle becomes stuck in a sandy patch with its wheels spinning. The vehicle must cope with this perturbation, perhaps by deflating its tires to get enough traction to move beyond the sandy area. Yet, due to the deflated tires, the vehicle must adjust its driving model, in particular, its steering on hard ground to avoid collisions. Throughout the vehicle's mission, it had to cope with unexpected events and changes, and the steps it took to cope with them were in the interest of its goal to reach its final destination, not in order to maintain a particular state or execute a procedure. This manuscript defines resilience for achievement-oriented agents, as exemplified by the autonomous vehicle, which are referred to hereafter as *goal-based agents*.

Throughout this manuscript, the term *algorithm* refers to the collection of decision-making processes that map sensed information to actions. The behavior of an agent's algorithm is determined by: the agent's configuration (sensors, effectors, memory, and computational resources) and the environment. For goal-based agents, the probability that an agent's behaviors achieve its goals is referred to as the agent's *competency* [16]. If the world can be modeled with high fidelity, then competency-related guarantees of stability, optimality, and efficiency can be made about the algorithm [17]. Unfortunately, competency is not sufficient to define an agent's success, because some aspects of the environment may not be anticipated at design time, and; therefore, are not sufficiently well-modeled to provide competency-related guarantees. *Resilience* also must be a design objective. Resilient goal-based agents react to perturbations to the environment or the agent itself in ways that enable the agent to achieve its goal.

Formal resilience definitions for goal-based agents are introduced, along with metrics that permit comparing different algorithms. The manuscript builds on the definitions from two conference papers and an extended abstract [16], [18], [19], making three substantial improvements: (1) the mathematical formalism is extended and refined, (2) the definitions of resilience and perturbation types are more precise and complete, and (3) the three case studies are novel. Most importantly, the three case studies illustrate in detail how to use the resilience framework's formal definitions to obtain theoretical results that can elucidate why some problems have tradeoffs between optimality and resilience, and in some cases even reveal the extent of the tradeoff. The case studies demonstrate how empirical measurements can be used based on the framework's definitions to understand an algorithm's resilience to various types of perturbations that in turn, can help algorithm designers determine which design choices are needed to reach their performance goals for the system (e.g. sacrificing optimality to achieve a certain level of resilience).

II. RELATED WORK

The definitions of resilience found in the literature are reviewed, followed by a discussion of useful metrics and measurement concepts derived from the definitions.

A. RESILIENCE DEFINITIONS

The first resilience category is a *state space-based* definition known as *engineering resilience* [13], which is closely related to the notion of *robustness* in control systems [3], [13], [20], [21]. The key concept is that a resilient system can maintain a desired property (e.g., stability), or return to a desired equilibrium state after a perturbation [3]. Engineering resilience is suited for systems with maintenance and procedural goals in which a system property must be maintained at an equilibrium point, but is insufficient for systems with achievement goals that require actions to bring about a desired state.

The second resilience category is also state space-based and referred to as *ecological resilience*. Ecological resilience applies to natural phenomena, including biological and network systems, that lack fixed stable points in their dynamics [22]–[24]. For example, predator-prey systems have predator and prey populations that vary over time. A resilient ecological system preserves the system's *identity* by ensuring relationships between system elements persist over time [13]. Ecological resilience assumes maintenance goals.

The third state-based category defines resilience using a *viability kernel* [25]. Viability assumes only that the system's goal is reachable and does not impose the existence of a baseline or equilibrium condition. Viability-based resilience frameworks exist for ecological systems [25]–[27] and show promise as a generic approach to measuring resilience for both maintenance and achievement goals. There are two primary differences between viability-based resilience and the notion proposed in this manuscript. First, viability-based resilience is only defined for threshold-based performance goals, while this manuscript's resilience framework introduces a much larger class of goals that can be defined via general goal functions. This manuscript also introduces resilience metrics based on the formal resilience concepts that can be used to measure how well an algorithm is achieving its goal under different perturbation types. Viability-based notions of resilience do not provide such metrics.

A fourth category, called *switching resilience*, is a qualitative framework that describes resilience as the ability to (re)establish stability, perhaps around a new equilibrium state, when new operating parameters or new goals arise [28]. Switching resilience allows transitions between different maintenance or procedural goals when the boundaries of a system's *competence envelope* are altered due to changing conditions. Switching resilience requires a system to change its competence envelope in order to continue operation, which may require the system to alter its goals or its structure. The introductory terrain traversal example's vehicle altered how it achieved its goal in order to cope with the environmental perturbations. While switching resilience can apply to agents with maintenance goals and those with achievement goals, it lacks a precise mathematical formulation with associated metrics. This manuscript addresses this limitation by making precise the high-level description of switching resilience.

Various designs or algorithms have been qualitatively described as improving resilience under various circum-

stances (e.g., [29]), but qualitative definitions typically do not yield generalizable quantitative metrics. Some approaches to resilience provide precise formulas for calculating resilience [30]–[32], but are restricted to specific algorithms or applications, and are not applicable in general to any system with sensors, actuators, and computational resources as in the notions of resilience defined in this manuscript.

B. RESILIENCE METRICS

One class of resilience metrics for natural and networked systems is based on susceptibility to disturbance or catastrophe, and are based on the absence of system fragility [24], [33]. Many more direct measures of resilience are intended for maintenance goals and are based on the so-called *resilience triangle* (e.g., [34]–[36]), which represents performance lost due to a perturbation and subsequently recovered, relative to a baseline value. Three different types of baselines exist: the magnitude of total change in a system variable relative to its pre-perturbation value, the percentage of recovery relative to its pre-perturbation value, and the percentage of recovery relative to the maximum change [37].

A number of metrics exist for characterizing the disturbance and subsequent restoration of the system's performance relative to a stable baseline [4], [5], [7], [10]. *Resistance*, the magnitude of the deviation between the equilibrium and current performance values, and *recovery*, the speed at which the system returns to equilibrium, provide insight into how well the system withstands disruption [38]. Resistance and recovery are not necessarily correlated [37], and systems with high resistance and low recovery differ from systems with low resistance and high recovery.

Resilience metrics often impose simplifying assumptions. The *triangle model* assumes that both the perturbation's effects and the performance recovery are relatively linear processes, and are treated as non-overlapping phases. Similarly, engineering resilience metrics often assume that the performance measure is constant in the absence of a perturbation. Metrics often examine the effect of perturbations on a single performance measure even when a system has multiple performance measures [36], which requires the overall system resilience to be inferred from the individual results.

Three quantifiable properties for assessing resilience in ecological systems are: *latitude*-the total amount a system can change before crossing the current basin of attraction's threshold, *resistance*-the difficulty in changing the system, and *precariousness*-how close the system currently is to a threshold [39]. Latitude is represented by the basin of attraction's width, resistance by the basin's depth, and precariousness by the minimum distance between the current system state and the edge of the current basin of attraction. Systems operating in different domains of attraction are not directly comparable, and comparing the value of a single property (e.g., latitude) provides no insight into the other properties. Consequently, metrics have largely been evaluated on a per-case basis, even though the description of ecological resilience has remained consistent [39]–[41]. The development and validation of a generalized framework for measuring ecological resilience remains an open problem [42]–[45].

Overall, these metrics do not apply to agents with achievement goals. While some metrics can measure resilience relative to maintenance goals, they do not apply to agents with achievement goals (e.g., there may simply be no recovery to a pre-perturbation value, as illustrated by the introductory example's vehicle deflating its tires). Similarly, the metrics for ecological systems are not applicable to agents with achievement goals where many of the system's initial properties may have irreversibly changed due to the system's attempt to overcome the changes caused by the perturbation (e.g., the introductory example's vehicle dropping its containers).

III. RESILIENCE FOR GOAL-BASED AGENTS

An agent's algorithm generates action commands that alter the world. Perturbations alter what actions an agent uses and how those actions affect the world. A resilient goal-based agent can accomplish its goal in the presence of perturbations.

A. AFFORDED BEHAVIOR POTENTIAL

Defining resilience requires understanding the effects of both agent behaviors and perturbations. The *world model* is a discrete-time, discrete-state tuple $\langle S, A, E \rangle$. S denotes the set of world states, and $s_t \in S$ denotes a finite n-element tuple of state variables at time t. A denotes the set of actions, a, that induce a transition from one state to another. E represents how *environment* states are related via actions, where

$$E \subseteq S \times A \times S.$$

Elements of the environment relation are present-state-action-next-state tuples, (s_t, a, s_{t+1}).

The environment relation is broad enough to include many common world models, including transition systems, difference equations, Markov processes, and other probabilistic systems. The present-state-action-next-state tuple, (s_t, a, s_{t+1}), abstracts many other representations for discrete time, discrete state systems. The relation makes no commitment to how these tuples are formed, but does make a commitment to a weak form of the Markov property where next state depends only on the current state and current action. The Markov property is "weak" because the environment relation does not specify conditional probabilities. Note that the environment relation allows transitions to next states from current states, both when an agent acts and in the absence of agent action. Thus, the environment relation is an *open system*, subject to both intentional and external state transitions.

A sequence of actions induces a corresponding sequence of transitions between states. A finite *state-action trajectory* with initial state s_0 is a t-step time-indexed history of states and actions, beginning at state s_0:

$$\begin{aligned} \xi(s_0, t) &= [(s_0, a_0), (s_1, a_1), \ldots, (s_{t-1}, a_{t-1}), s_t] \\ &\in \left[\prod_{\tau=0}^{t-1} (S \times A)\right] \times S. \end{aligned} \tag{1}$$

Let $\mathbb{T}(s_0, t)$ denote the set of all t-step trajectories that can occur in the world when no restrictions are placed on what action sequences can be performed,

$$\mathbb{T}(s_0, t) = \{\xi(s_0, t) : \xi(s_0, t) \text{ is possible in } E\}.$$

The world's *afforded behavior potential* given time bound $T_{\max}$ represents all possible trajectories from any initial state,

$$\mathbb{B}(T_{\max}) = \bigcup_{s \in S} \bigcup_{t \leq T_{\max}} \mathbb{T}(s, t).$$

An embodied agent usually does not have enough control authority, observational power, or computational resources to choose actions that reach the world's afforded behavior potential. Indeed, a goal-based agent seeks to induce only those trajectories that achieve its goal.

B. ALGORITHM BEHAVIOR POTENTIAL

Three practical restrictions exist when the algorithm is part of an embodied agent:

1) The agent may observe only a portion of the state due to sensor limitations, $S_{\text{obs}} \subseteq S$. An element of the observable subspace at time t is denoted by s_t^{obs}.
2) The agent may influence only a portion of the action space, $A_{\text{eff}} \subseteq A$. An element of the action space that can be effected by the agent at time t is denoted by a_t^{eff}.
3) Bounded memory and bounded computation time limit what an agent can do in the world. Consequently, computational resources are part of the state space, $S_{\text{comp}} \subseteq S$. The computational state $s_t^{\text{comp}} \in S_{\text{comp}}$ is understood in the Turing machine sense to be what is needed by the algorithm to make future decisions (e.g., an estimate of current environment state that the algorithm stores for the next time step).

Represent the agent's algorithm as a relation:

$$X \subseteq S_{\text{obs}} \times S_{\text{comp}} \times A_{\text{eff}} \times S_{\text{comp}}.$$

Elements of the algorithm relation are tuples $[(s^{\text{obs}}, s^{\text{comp}}), (a^{\text{eff}}, s^{\text{comp}'})]$. The state available to the algorithm is $(s^{\text{obs}}, s^{\text{comp}})$, which is composed of the sensor input and the agent's current internal computational state. The product $S_{\text{obs}} \times S_{\text{comp}}$ is almost always a proper subset of the state space for real-world systems, because real-world sensors and finite memory are insufficient to capture the world perfectly [46]. The algorithm's output $(a^{\text{eff}}, s^{\text{comp}'})$ is the action the agent will perform and an updated computational state. The algorithm relation X is functionally equivalent to a "state table" defining the possible actions an agent can take in a given state. Representing the algorithm as a relation means that multiple actions can be associated with a given state.

The environment relation, E, and algorithm relation, X, can be combined to create the trajectories an algorithm can induce in an environment. Since $S_{\text{obs}} \times S_{\text{comp}} \subseteq S$, the present state of the world is composed of the algorithm's present state and any additional information unavailable to the agent, $s_t = [(s_t^{\text{obs}}, s_t^{\text{comp}}), \tilde{s}_t]$ where $\tilde{s}_t$ represents everything in the world state that is not part of the agent's state. $A_{\text{eff}} \subseteq A$, the effectual action taken in the world can be written as $a_t = [a_t^{\text{eff}}, \tilde{a}]$, where $\tilde{a}_t$ represents everything that is not explicitly part of the agent's action.

A state-action trajectory, $\xi(s_0, t)$, is created for a starting state, s_0, using Algorithm 1. Line 3 finds all tuples that have

Algorithm 1 Construct a t-step trajectory from an initial state s_0.

Require: initial state s_0 and time $t - 1$

1: $\xi \leftarrow []$ and $s \leftarrow s_0$ ▷ $s = [(s^{\text{obs}}, s^{\text{comp}}), \tilde{s}]$
2: **for** $\tau \in [1, 2, \ldots, t]$ **do**
3: Select $[(s_\tau^{\text{obs}}, s_\tau^{\text{comp}}), a_\tau^{\text{eff}}, s_\tau^{\text{comp}'}]$ from X where $s_\tau = [(s_\tau^{\text{obs}}, s_\tau^{\text{comp}}), \tilde{s}_\tau]$
4: Select $(s_\tau, a, s_{\tau+1})$ from E where $a_\tau = (a_\tau^{\text{eff}}, \tilde{a}_\tau)$
5: $\xi \leftarrow [\xi, (s_\tau, a_\tau)]$
6: $\xi \leftarrow [\xi, s_t]$

a present state compatible with the agent's observed and computational state, and picks one of the tuples. Line 4 looks up all world present-state, actions, and next-states that have an action compatible with the agent's action from line 3. Line 5 appends the world state-action tuple to the trajectory, and line 6 appends the final state to the trajectory.

Algorithm 1 can generate multiple trajectories from a given start state, since both E and X allow non-determinism. Let $\mathbb{T}_X(s_0, t)$ denote the set that holds all t-step state-action trajectories produced for a given initial state (s_0) by algorithm X,

$$\mathbb{T}_X(s_0, t) = \{\xi(s_0, t) : \xi(s_0, t) \text{ is generated by } X \text{ in } E\}.$$

The dependency of $\mathbb{T}_X(s_0, t)$ on the environment is omitted from the notation when it is clear which environment is used.

State $s_0 = [s_0^{\text{obs}}, s_0^{\text{comp}}, \tilde{s}_0]$ is a *valid* initial state if the agent's initial state is $(s_0^{\text{obs}}, s_0^{\text{comp}})$. Let S_{init} represent the set of valid initial states. An *algorithm's behavior potential* represents the agent's bounded ability to effect change in the world, and is defined as the set of possible trajectories generated from any valid initial state:

$$\mathbb{B}_X(T_{\max}) = \bigcup_{s_0 \in S_{\text{init}}} \bigcup_{t \leq T_{\max}} \mathbb{T}_X(s_0, t).$$

In summary, $\mathbb{T}_X(s_0, t)$ is the set containing all t-step trajectories that can be produced from a given starting point. A state-action trajectory $\xi(s_0, t) \in \mathbb{T}_X(s_0, t)$ is a sample from the possible trajectories that can be produced by an algorithm. The algorithm's behavior potential, $\mathbb{B}_X(T_{\max})$ is the set of all trajectories that the algorithm can produce from any valid initial state, given a time bound $T_{\max}$.

C. COMPETENCY

Defining resilience requires understanding how *competently* an algorithm satisfies an achievement goal in the presence of perturbations. Formally, an autonomous agent is *competent* if it has an algorithm that induces state-action trajectories that satisfy the goal with sufficiently high probability within

a given time period. Competency requires a probability, P, defined for each trajectory set in the afforded behavior potential $\mathbb{B}$. Assume (with little loss of generality) that all sets of trajectories are measurable with respect to probability P.

Represent a goal, G, as a Boolean random variable defined with respect to the probability space. A Boolean random variable is very general and includes tests of single states (e.g., the agent reached a desired state), tests of state trajectories only (e.g., a trajectory avoided a danger region), goals that include valuations over actions (e.g., maximum torque limits were not exceeded), goals that compare performance measures to acceptable performance thresholds (e.g., the trajectory's score was high enough), or combinations of these tests. Let $\mathbb{S}_0$ denote the initial state random variable, defined with respect to the same probability space. The *derived distribution* of achieving the goal given the start state is:

$$P_{G|\mathbb{S}_0}(\texttt{true}|s_0;t,X) = P(\{\xi(s_0,t) \in \mathbb{B}_X(T_{\max}) : G(\xi(s_0,t)) = \texttt{true}\}).$$

The conditioning bar indicates that the goal random variable is conditioned on the starting state random variable, and the semicolon indicates that time t parameterizes the conditional probability. The derived distribution $P_{G|\mathbb{S}_0}(\texttt{true}|s_0;t,X))$ represents the measure of the set of all t-step trajectories that are produced by the algorithm and that satisfy the goal.

Formally, algorithm X *competently accomplishes goal* G *given initial state* s_0 if

$$P_{G|\mathbb{S}_0}(\texttt{true}|s_0;t,X) \geq \theta \tag{2}$$

for the given parameters t and θ. Note that the time bound $T_{\max}$ is implicit in Equation (2). In other words, an agent competently accomplishes a goal in time t if the set of goal-satisfying trajectories from a given start state have a probability measure that exceeds θ.

A time-bound, tolerance, and starting state are all necessary, because it is not always possible for an agent to perfectly accomplish its goal in a reasonable time from any initial state in many real environments. The start state is a formal expression of where the agent begins. The time bound $T_{\max}$ represents the idea that a goal must be achieved in a finite time, after which the goal is no longer worth pursuing. The probability threshold θ is a formal expression of the tolerance associated with achieving a goal (i.e., it represents the "size" of the *competency envelope* [28]).

Defining competency as a logic-based goal function permits a rich goal set, but some goal types do not naturally fit the logic-based definition. An example incompatible goal is "an agent is competent if it uses the optimal solution." *Optimality*, means that an algorithm is objectively superior to all other algorithms according to a performance criterion. Consequently, optimality defines competence relative to the set of all possible algorithms and the objective function, instead of evaluating whether trajectories satisfy a goal. Another example is "operate as long as possible," where competence is defined with respect to a time value that cannot be specified precisely. Each example has a related boolean goal: "an agent is competent if a numerical score over its trajectories exceeds a desirable value," and "an agent is competent if it can operate at least T units of time into the future."

D. PERTURBATIONS

Perturbations affect what an agent observes, what actions are possible, how the environmental states change, and what an agent can compute. Without loss of generality, assume that perturbations to the probability measure P alter the frequency with which trajectories are produced and not the underlying sample space. Thus, perturbations affect an algorithm's competency if they alter either the algorithm's behavior potential $\mathbb{B}_X$, or the probability of trajectories within the behavior potential P. Let $B_X^{\mathcal{I}}$ (defined below) represent the perturbed set of trajectories. Formally, a perturbation occurs when

$$\left(\mathbb{B}_X^{\mathcal{I}} \neq \mathbb{B}_X\right) \vee \left(\exists \xi \in \mathbb{B}_X^{\mathcal{I}} \text{ and } \exists \delta_i \in \mathcal{I}, \text{ such that } P(\{\xi\}|\delta_i) \neq P(\{\xi\})\right).$$

Represent changes to an algorithm's behavior potential using an indexed set of *perturbation mappings*, δ_i, where i is an element of some finite index set $\mathcal{I}$. Each perturbation mapping changes the set of trajectories $\mathbb{T}_X(s_0)$ into a new set of trajectories $\delta_i(\mathbb{T}_X(s_0))$. For simplicity, assume that all sequenced combinations of perturbations are included in $\mathcal{I}$. Formally, an *algorithm's perturbed behavior potential* is defined as the set of all possible trajectories:

$$\mathbb{B}_X^{\mathcal{I}}(T_{\max}) = \bigcup_{s \in S_{\text{init}}} \bigcup_{t \leq T_{\max}} \bigcup_{i \in \mathcal{I}} \delta_i(\mathbb{T}_X(s,t)).$$

$\mathbb{B}_X^{\mathcal{I}}(T_{\max})$ contains all trajectories from any initial state with any perturbation.

Five *perturbation function types* exist: ablation, addition, distortion, shift, and transformation. The ablation and addition types change the set of trajectories by changing the set of states, actions, or tuples in the environment, or the agent's algorithm. Distortion types change the trajectory probability. Shifts blend ablation, addition, and distortion types, while transformations alter goals. Let the superscript $'$ indicate a change caused by a perturbation.

Ablation occurs when there is loss of information, control, or possible trajectories as a result of changes to the action space, state space, or environment. An ablation occurs if:

$$\begin{aligned} \text{Ablation}(A) &= A', \text{ where } A' \subset A, \\ \text{Ablation}(S) &= S', \text{ where } S' \subset S, \text{or} \\ \text{Ablation}(E) &= E', \text{ where } E' \subset E. \end{aligned}$$

Ablations remove tuples from the algorithm's behavior potential. Example ablation events include: 1) losing an effector, 2) losing a sensor, 3) losing a computational resource, and 4) an environmental change that reduces what can be done.

Addition occurs when there is an increase in information, control, or possible trajectories as a result of changes to the

action space, state space, or environment. Addition is the inverse of ablation, and is represented as:

$$\begin{aligned}\text{Addition}(A) &= A', \text{ where } A' \supset A,\\ \text{Addition}(S) &= S', \text{ where } S' \supset S, \text{or}\\ \text{Addition}(E) &= E', \text{ where } E' \supset E.\end{aligned}$$

Addition perturbations add trajectories to the algorithm's behavior. Example addition events include: 1) an obstacle is removed, 2) the agent acquires a new sensor or a new computational resource, and 3) a broken effector is repaired.

Distortion occurs when the probability of a trajectory is altered without altering the algorithm's behavior potential. Examples of distortion events include: 1) a sensor degrades, 2) a malfunctioning effector begins to work, or 3) a software defect occurs that reduces memory utilization.

Shifts occur when multiple ablations, additions, and distortions occur. Shifts combine the effects of multiple ablation, addition, or distortion instances, and can affect combinations of states, actions, state-action transitions, or probabilities.

Transformations occur when the goal is changed. Examples of a transformation include: 1) a human manager changes the agent's goal, or 2) an agent endowed with self-assessment abilities determines that it can no longer competently achieve an existing goal and selects a different goal to pursue. Transformations are beyond the scope of this manuscript.

E. RESILIENCE DEFINITION AND METRICS

Even when the environment or the algorithm is perturbed, a resilient algorithm can still observe enough of the world, has enough control authority, and possesses sufficient computational resources to accomplish the goal with high probability within a reasonable time bound. Formally, an agent is resilient to a perturbation δ if

$$P_{G|\mathbb{S}_0}(\texttt{true}|s_0,\delta;t,X) \geq \theta. \tag{3}$$

Note that resilience is relative to the goal being pursued and the specific perturbation that occurs. For example, a robot navigating a previously mapped environment can exhibit resilience to newly discovered obstacles by replanning its route, but may not be resilient if its sensors are noisy, inhibiting its ability to model the obstacles' locations accurately.

Equation (3) defines resilience as a binary decision problem, which does not allow different algorithms to be usefully compared. The remainder of this section introduces three resilience metrics for comparing algorithms: *dominance*, *power*, and *efficiency*. The metrics will be defined for a single start state. Generalizing to the multiple possible starting states is left to future work. Comparing algorithms is fair only if they operate in the same environment, so the same environment is assumed for both algorithms.

Consider algorithms X_m and X_n, and consider perturbation δ. Algorithm X_m *weakly dominates* algorithm X_n for initial state s_0 and environment E if

$$\forall t \leq T_{\max}\ P_{G|\mathbb{S}_0}(\texttt{true}|s_0,\delta;t,X_m) \geq P_{G|\mathbb{S}_0}(\texttt{true}|s_0,\delta;t,X_n).$$

X_m weakly dominates X_n if the success probability of X_m is always at least as high as X_n. Weak dominance can be used to partially order different algorithms, $X_m \succeq^D X_n$, if and only if X_m weakly dominates X_n, where the superscript D indicates that the preference ordering is derived from dominance. Since dominance does not always apply, it induces only a partial order. The notation $X_m \succeq^D X_n$ omits the dependence on the starting state for readability. Dominance is useful in detecting whether one algorithm is clearly inferior to another.

The second resilience comparison metric is time *efficiency*, which is defined using the time required for an algorithm to satisfy a performance threshold

$$t_{\text{sat}}(X;\theta,s_0,\delta) = \arg\min_{0\leq t\leq T_{\max}} \Big(P_{G|\mathbb{S}_0}(\texttt{true}|s_0,\delta;t,X) \geq \theta\Big).$$

If there does not exist a time for which the algorithm is competent in the presence of a perturbation, then $t_{\text{sat}}(X;\theta,s_0,\delta) = T_{\max}$. Efficiency, $\mathcal{F}$, is defined using the time-bound

$$\mathcal{F}_{\text{sat}}(X;\theta,s_0,\delta) = \frac{T_{\max} - t_{\text{sat}}(X;\theta,s_0,\delta)}{T_{\max}},$$

which equals one if the threshold is immediately satisfied, and equals zero if the threshold is not satisfied before the time bound is reached. When the algorithm, starting state, and perturbation are unambiguous, it is useful to represent efficiency by $\mathcal{F}(\theta)$ to make the performance threshold explicit.

The efficiency metric induces a total order over all algorithms, where X_m is at least as efficient as algorithm X_n, given an initial state and perturbation, which is written as

$$\begin{aligned}&\Big(X_m \succeq^{\mathcal{F}(\theta)} X_n\Big)\\ &\quad\Leftrightarrow \Big(\mathcal{F}_{\text{sat}}(X_m;\theta,s_0,\delta) \geq \mathcal{F}_{\text{sat}}(X_n;\theta,s_0,\delta)\Big) &(4)\\ &\quad\Leftrightarrow \Big(t_{\text{sat}}(X_m;\theta,s_0,\delta) \leq t_{\text{sat}}(X_n;\theta,s_0,\delta)\Big) &(5)\\ &\quad\Leftrightarrow \Big(t_{\text{sat}}(X_m;\theta,s_0,\delta) - t_{\text{sat}}(X_n;\theta,s_0,\delta)\Big) \leq 0. &(6)\end{aligned}$$

Equation (5) shows that the ordering does not depend on $T_{\max}$, but with an important caveat. Neither X_n nor x_m may meet the performance threshold θ when T_{max} is low, which means that t_{sat} is the same for both algorithms and $\mathcal{F}_{sat}$ is zero for both algorithms. This scenario results in X_m and X_n being equivalent. However, when T_{max} is increased one algorithm may meet the threshold while the other fails to, which means that one algorithm has an efficiency of zero and the other has a positive efficiency. For this scenario, one algorithm is strictly better than the other. The ordering therefore must use a weak ordering $\succeq$ instead of a strict ordering $\succ$. Stated simply, when T_{max} is small, the efficiency metric may not be able to differentiate between algorithms. Equation (6) represents the ordering as a difference, which is a simple test used in the third case study.

The third resilience comparison metric is the *power* of an algorithm, defined as the *peak success probability* achieved before a time deadline,

$$\mathcal{P}_{\text{peak}}(X;t,s_0,\delta) = \max_{0\le\tau\le t} P_{G|\mathbb{S}_0}(\texttt{true}|s_0,\delta;\tau,X)$$
$$= P_{G|\mathbb{S}_0}(\texttt{true}|s_0,\delta;t,X),$$

where the second equality holds if and only if the success probability is a nondecreasing function of time (e.g., when nothing can "undo" the goal once it is achieved). When the algorithm, starting state, and perturbation are unambiguous, it is useful to represent power by $\mathcal{P}(t)$ to make the time explicit. Applying the definition of $\mathcal{P}$ to algorithms X_m and X_n allows the power metric to induce a total order over all algorithms,

$$\left(X_m \succeq^{\mathcal{P}(t)} X_n\right)$$
$$\Leftrightarrow \left(\mathcal{P}_{\text{peak}}(X_m;t,s_0,\delta) \ge \mathcal{P}_{\text{peak}}(X_n;t,s_0,\delta)\right) \quad (7)$$
$$\Leftrightarrow \left(P_{G|\mathbb{S}_0}(X_m;t,s_0,\delta) \ge P_{G|\mathbb{S}_0}(X_n;t,s_0,\delta)\right),$$
$$\text{when } P \text{ is nondecreasing} \quad (8)$$
$$\Leftrightarrow \left(P_{G|\mathbb{S}_0}(X_m;t,s_0,\delta) - P_{G|\mathbb{S}_0}(X_n;t,s_0,\delta)\right) \ge 0. \quad (9)$$

There are two ways to define resilience when multiple perturbations are possible. The first is to evaluate dominance, efficiency, and power on a case-by-case basis. This approach allows conclusions of the form "X_n is more powerful/efficient than X_m for perturbation δ_j, but less powerful/efficient for perturbation δ_k." The second approach to multiple perturbations is to aggregate the resilience metrics across perturbations using worst-case or average-case analysis. Worst-case resilience metrics can be computed over the set of possible perturbation mappings, and average values can be computed over a probability distribution of perturbation functions.

The worst-case efficiency and power values for algorithm X are given by:

$$\lfloor\mathcal{F}_{\text{sat}}\rfloor(X;\theta,s_0) = \min_{i\in\mathcal{I}}\{\mathcal{F}_{\text{sat}}(X;\theta,s_0,\delta_i)\},$$
$$\lfloor\mathcal{P}_{\text{peak}}\rfloor(X;t,s_0) = \min_{i\in\mathcal{I}}\{\mathcal{P}_{\text{peak}}(X;t,s_0,\delta_i)\},$$

respectively. Average-case values depend on the probability of the various perturbation functions. Let $\mathcal{I}$ denote a random variable over the set of all possible perturbation functions, and let $P_D(\delta_i)$ denote the corresponding derived probability distribution. Let $\mathbb{E}_{P_D}$ represent the expected value operator, where expectation is with respect to the distribution $P_D(\delta_i)$. The average-case efficiency and power values are:

$$\overline{\mathcal{F}_{\text{sat}}}(X;\theta,s_0) = \mathbb{E}_{P_D}\mathcal{F}_{\text{sat}}(X;\theta,s_0,\delta_i),$$
$$\overline{\mathcal{P}_{\text{peak}}}(X;t,s_0) = \mathbb{E}_{P_D}\mathcal{P}_{\text{peak}}(X;t,s_0,\delta_i),$$

respectively. Algorithms X_m and X_n can be ordered using the worst-case efficiency and power values,

$$\left(X_m \succeq^{\lfloor\mathcal{F}(\theta)\rfloor} X_n\right) \Leftrightarrow \left(\lfloor\mathcal{F}_{\text{sat}}\rfloor(X_m;\theta,s_0) \ge \lfloor\mathcal{F}_{\text{sat}}\rfloor(X_n;\theta,s_0)\right),$$
$$\left(X_m \succeq^{\lfloor\mathcal{P}(t)\rfloor} X_n\right) \Leftrightarrow \left(\lfloor\mathcal{P}_{\text{peak}}\rfloor(X_m;t,s_0) \ge \lfloor\mathcal{P}_{\text{peak}}\rfloor(X_n;t,s_0)\right),$$

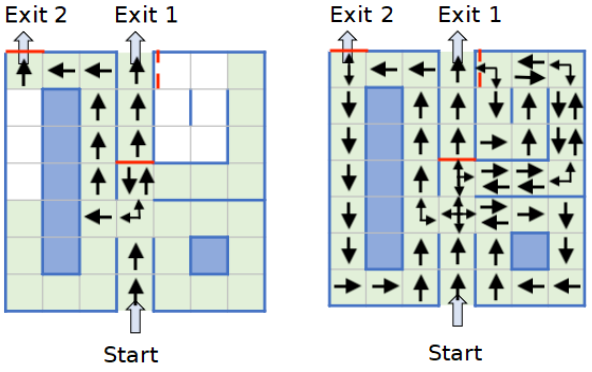
respectively, where the superscripts $\lfloor\mathcal{F}(\theta)\rfloor$ and $\lfloor\mathcal{P}(t)\rfloor$ indicate that the preference is derived from the worst case efficiency and the worst case power. Orderings using average-case efficiency and power are defined similarly.

IV. CASE STUDY: MEMORY-CONSTRAINED NAVIGATION

A robot designer wants to analyze the resilience of two algorithms. The designer identifies the world, algorithms, behavior potentials, and perturbations. The algorithms' resilience properties can be compared using power and efficiency.

A. ENVIRONMENT AND PERTURBATIONS

The agent is a simulated four-wheeled robot equipped with GPS and lidar that must travel through the maze shown in Fig. 1. The figure shows that the environment E is a two-dimensional grid where the blue lines represent walls through which the robot cannot pass. The robot can move one square per time step. The robot's available abstract perceptions (S_{obs}) are grid cells, adjacent walls, and intersections.



(a) X_{DFS} (b) X_{Trem}

FIGURE 1: The map of the environment that the robot must traverse. Solid red lines are obstacles that can appear, and the dashed red line is a door that can open. The shaded squares represent information described in Section IV-C about two memory-limited algorithms: (a) X_{DFS} and (b) X_{Trem}. White squares are not reachable by X_{DFS}.

Robot actions are north, south, east, and west (A_{eff}), which are implemented by low-level rotational and translational controllers. The computational states (S_{comp}) are algorithm parameters, bookkeeping variables, and a stack that has only ten slots. The robot's goal, G, is to traverse from the marked start location (*Start*) to one of the exit locations (*Exit 1* or *Exit 2*). The solid and dashed red lines in Fig. 1 represent perturbations. The solid red line near the center of the map represents a roadblock (an *ablation perturbation*) that blocks the direct route to *Exit 1*, the solid red line across *Exit 2* represents a roadblock across the exit, and the red dashed line represents a door that opens (an *addition perturbation*) that provides a new route to an exit.

B. ALGORITHMS

The first algorithm's, X_{DFS}, planner uses the depth-first-search (DFS) given in Algorithm 2. The algorithm uses the ten-element stack to invoke recursion. The DFS returns success if an exit is reached (line 3), failure if the stack is full (line 4), or otherwise searches through each action (line 5). The NextCell method returns "wall" if the action leads to a wall collision, otherwise it returns the cell induced by the action (line 7). A wall terminates the recursion with a failure (line 8), otherwise the DFS method is recursively called (line 9). If the recursion ends with success, then the recursion unwinds and saves the sequence of actions (lines 9–10).

Algorithm X_{DFS} uses planning to find optimal (shortest length) paths. The path is executed by following the actions in order. If a wall is found (or not found) by the robot in a location that did not match the map used by the planner, then the DFS method is called again, but with a constraint. The constraint is that the actions executed to bring the robot to the new (missing) wall are used to initialize the stack (replacing line 1 of Algorithm 2), which allows replanning while retaining the ability to find optimal paths.

Algorithm 2 X_{DFS}

```
1:  initialize path as an empty vector
2:  procedure DFS(s^current)
3:      if s^current == Exit then return(success)
4:      else if stack.status == full then return(failure)
5:      else
6:          for a ∈ {N, S, E, W} do
7:              s^next ← NextCell(a)
8:              if s^next == wall then return(failure)
9:              else if DFS(s^next) == success then
10:                 path.append(a)
11:                 return(success)
```

Algorithm 3 X_{Trem}

```
1:  initialize empty stack
2:  procedure Trem()
3:      while s^current ≠ exit do
4:          if IsDeadEnd(s^obs) then s^next ← s^previous
5:          else if s^current ≠ intersection then s^next ← NextCell(s^current, s^previous)
6:          else s^next ← IntersectionManager(s^current)
7:          [s^previous, s^current] ← MoveTo(s^next)
8:      return(success)                                   ▷ Reached exit
9:  procedure IntersectionManager(s^current, s^previous)
10:     if first time in s^current then
11:         if stack.status == full then return(s^previous)   ▷ Turn around
12:         else s^enter ← s^previous and stack.push(s^current)
13:     s^next ← GetFreeNeighbor(s^current, s^arrive)
14:     if s^next == None then stack.pop() and s^next ← s^enter
15:     return(s^next)
```

The second algorithm, X_{Trem}, is a version of Tremaux's algorithm, where the robot moves while searching for an exit. X_{Trem} uses the 10-element stack to track intersections as they are explored. The Trem main method, given in Algorithm 3, succeeds when the current state is an exit (lines 3 and 8). The algorithm sends the robot back the way it came if the robot encounters a dead-end (line 4). The robot proceeds through a cell to the next open cell if the cell is not at an intersection (line 5). Otherwise, the intersection stack manager determines the next state (line 6). The algorithm uses the MoveTo method to select the appropriate action from $\{N, S, E, W\}$, take the action, and update the next state (line 7).

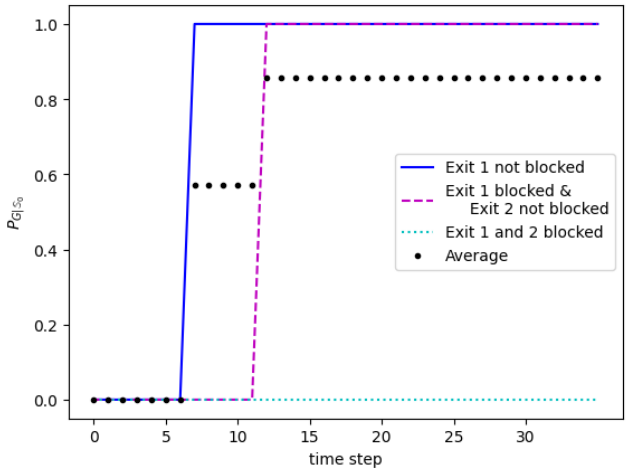
The IntersectionManager method first checks whether the intersection has been visited or not (line 10). Two things occur when the intersection is visited: first, the robot turns around if the stack is full (line 11); and second, the algorithm notes how the robot entered the intersection and pushes the intersection onto the stack. The algorithm uses GetFreeNeighbor function to find a cell adjacent to the intersection that has not been visited (line 13). If there are no adjacent cells to explore, then the algorithm removes the intersection from the stack and tells the robot to exit the intersection the same way it first entered (line 14). The algorithm then returns the next cell.

C. TRAJECTORIES AND PERTURBED ALGORITHM BEHAVIOR POTENTIAL

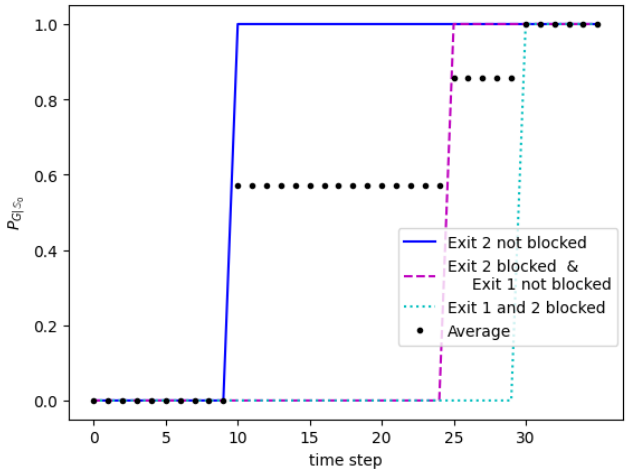
The perturbed behavior potentials, $\mathbb{B}_X^{\mathcal{I}}$, for the two algorithms are shown in Fig. 1a. The behavior potential for the deterministic algorithm X_{DFS} is shown on the left as shaded cells with directional arrows. The directional arrows indicate trajectories that are produced depending on what perturbations occur. If the path to *Exit 1* is not blocked, then the robot moves directly to the exit. If the path to *Exit 1* is blocked, but *Exit 2* remains open, then the robot travels to the obstacle, discovers that it is blocked, replans, and turns around to take the path to *Exit 2*. If the direct path to both exits is blocked, the robot cannot find any exit, even if the door in the upper right of the maze opens. The shaded regions without arrows indicate cells that are considered by X_{DFS}, but are not part of any feasible path produced by the algorithm.

X_{Trem} can move the robot to any cell on the map, because no path contains more than 10 intersections, as shown by the arrows connecting the shaded cells in Fig. 1b. X_{Trem} chooses next cells in the order W, N, S, E, which was chosen to illustrate differences between the algorithms. When *Exit 2* is not blocked, the robot goes west at the first intersection and finds the exit in 10 steps. If *Exit 2* is blocked, then the robot continues down the left corridor, returns to the first intersection, and tries to go north. If *Exit 1* is not blocked then the robot finds the exit in 25 steps, but if *Exit 1* is blocked and the door is open, then the robot finds the exit in 31 steps. All possible trajectories can be found by following the arrows in the figure. The set of possible trajectories includes trajectories that arise only when one exit is blocked, both exits are blocked, and both exits are blocked with the door closed.

D. DOMINATION, EFFICIENCY, AND POWER



(a) X_{DFS}



(b) X_{Trem}

FIGURE 2: The probability of success for perturbation combinations with (a) the DFS and (b) Tremaux's algorithms.

Fig. 2 plots success probability $P_{G|\mathbb{S}_0}(\texttt{true}|s_0, \delta; T, X)$ as a function of time T for both X_{DFS} and X_{Trem}. X_{DFS} is designed to move toward *Exit 1* first, and X_{Trem} is designed to move toward *Exit 2* first. The solid blue line, dashed magenta line, and dotted cyan line indicate sets of possible perturbations δ. Note that the perturbations shown in Fig. 2(a) are organized to emphasize the effect of blocking *Exit 1* on the optimal algorithm X_{DFS}, and the perturbations shown in Fig. 2(b) are organized to emphasize the effect of blocking *Exit 2* on X_{Trem}.

There are seven conditions for which there is a feasible path to an exit: no perturbation, only *Exit 1* is blocked, only *Exit 2* is blocked, *Door* is open, and all combinations of these perturbations, except for the case when *Exit 1* and *Exit 2* are both blocked and the *Door* remains shut. This last perturbation is catastrophic in the sense that no algorithm can satisfy the goal. Assume that each condition is equally probable, and consider average efficiency and average power. The thick dotted black line in Fig. 2 shows the average success probability as a function of time.

Dominance. Neither algorithm dominates the other. Dominance only occurs if the probability of success of one algorithm is greater than or equal to the probability of success for the other for all time.

Worst Case Efficiency. Execution time in the Fig. 1 world equates to the number of cells the robot visited. Set $T_{\max} = 49$ is the length of the path that touches each cell in the world. X_{DFS} is designed to be efficient, because it seeks the shortest path. Worst case efficiency is determined by the minimum of the success probability curves over the possible perturbations. The worst case efficiency for X_{DFS} is zero for all positive performance thresholds, but the worst case efficiency for X_{Trem} is $\frac{49-31}{49} = 37\%$ for all positive performance thresholds.

Worst Case Power. Worst case power is determined by the minimum of the success probability curves over the possible perturbations. The worst case power for X_{DFS} is zero for all time, and the worst case power for X_{DFS} is one for $t \geq 31$.

Average Efficiency and Power. Fig. 2 shows that for any performance threshold less than $\theta = 0.85$, X_{DFS} is more efficient, because it reaches the threshold sooner than X_{Trem}. Similarly, up until time $t = 31$, X_{DFS} is more powerful, because it reaches the threshold sooner than X_{Trem}. However, X_{DFS} is not able to find a solution when both obstacles appear and the door is open. Thus, X_{Trem} is more efficient for a performance threshold higher than about $\theta = 0.85$, and X_{Trem} also has higher power after $t = 31$.

E. DISCUSSION

This memory constrained navigation case study demonstrates that the resilience metrics provide useful insight into the relative resilience properties of two memory-limited algorithms. An important insight is that the most resilient algorithm depends on performance tolerances and allowed time. Determining which algorithm is more efficient, using average efficiency, depends on performance tolerance, because high tolerance means that a fast, but low-performing algorithm can reach the threshold quickly. Similarly, which algorithm is more powerful, using either worst-case or average power, depends on the time at which power is measured. If more time is available, then X_{Trem} is more powerful. Moreover, this problem exhibits a tradeoff between efficiency and power, where an algorithm that produces more efficient solutions for some types of perturbations is less powerful for other types of perturbations, given enough time.

V. CASE STUDY: THEORY

The concepts and definitions introduced in Section III provide a general framework for determining and quantifying an agent's level of resilience. This case study shows how additional problem assumptions and constraints lead to insight-giving theoretical results.

A. ACTING IN AN OBSERVABLE AND DETERMINISTIC WORLD

Suppose an additional structure is added so that the environment relation E is given by a deterministic state transition system, $S \times A \to A$, with transition matrix M. Also suppose a fully observable world, $S_{\text{obs}} = S$, so that the algorithm relation X is represented as a policy mapping S to an action. Next, suppose the goal is satisfied if the world's state is within some proper subset of the world states, $S^* \subset S$, where S^* denotes the set of goal states. Suppose that each goal state is an *absorbing state*, meaning once an agent reaches the goal state no action causes it to move from that state. Finally, suppose the world is fully controllable, $A_{\text{eff}} = A$, so that the agent's actions uniquely determine the next state of the environment.

B. ALGORITHMS

Consider two different algorithm types, one deterministic and the other probabilistic, which are denoted by X_{det} and X_{prob}, respectively. The deterministic algorithm uses a *lookup table* specifying a unique action for each state, $\ell : S \to A$. The output of the table lookup is the action that moves the agent along the shortest path to a goal state. The probabilistic algorithm, X_{prob}, uses a *probability table* to specify the action probability given a state, $\pi(a|s) = P(a|s)$. The action with highest probability is the action that moves the agent to the state on the shortest path to the nearest goal, and the remaining probability is uniformly distributed across all other actions. The computational subspaces, S_{comp}, for the deterministic and probabilistic algorithms are ℓ and π, respectively.

Each algorithms' behavior potential can be represented as a directed graph, G_{det} and G_{prob}, respectively. The vertices in each graph represent states, $V_{\text{det}} = V_{\text{prob}} = S$. The directed edges in G_{det} represent the state transition for the action from the table lookup, $e(s_j, s_k) \in E_{\text{det}}$, if and only if $s_k = M(s_j|\ell(s_j))$. The outdegree of each vertex is one because the table lookup function ℓ is deterministic. The directed edges in G_{prob} also represent the state transition for any action with nonzero probability from the transition table, $e(s_j, s_k) \in E_{\text{prob}}$, if and only if there exists an a such that $\pi(a|s_j) > 0$, and $s_k = M(s_j|a)$. The outdegree of each vertex equals the size of the action space.

C. DOMINANCE IN THE ABSENCE OF PERTURBATION

The proof of Theorem 1 follows directly from the definitions of dominance.

Theorem 1. *For all starting states and in the absence of a perturbation, the deterministic algorithm dominates the probabilitistic algorithm,* $X_{\text{det}} \succeq^D X_{\text{prob}}$.

Proof. Recall from Section III-E that dominance in the absence of a perturbation means that $\forall t\ P_{G|\mathbb{S}_0}(\texttt{true}|s_0; t, X_{\text{det}}) \geq P_{G|\mathbb{S}_0}(\texttt{true}|s_0; t, X_{\text{prob}})$. The deterministic algorithm optimally chooses the action on the shortest path to a goal state. Let $T^*(s_0)$ denote the number of steps on the shortest path from s_0 to a goal state. Therefore, the success probabilities for the algorithms are

$$P_{G|\mathbb{S}_0}(\texttt{true}|s_0; t, X_{\text{det}}) = \begin{cases} 0 & t < T^*(s_0) \\ 1 & t \geq T^*(s_0) \end{cases},$$

$$P_{G|\mathbb{S}_0}(\texttt{true}|s_0; t, X_{\text{prob}}) = \begin{cases} 0 & t < T^*(s_0) \\ \alpha \in (0,1) & t \geq T^*(s_0) \end{cases}.$$

Stated simply, the deterministic algorithm succeeds precisely when the elapsed time equals the time to the nearest goal. The probabilistic algorithm can succeed no faster than the minimum time, but does not always succeed immediately at the minimum time, because probabilistic choices by the agent will sometimes deviate from the optimal path. □

D. DOMINANCE IN THE PRESENCE OF PERTURBATION

Consider an ablation, δ_{edge}, that removes the same single edge from both G_{det} and G_{prob}. Restrict attention to non-degenerate cases where (a) all vertices in G_{det} are not on the same shortest path, and (b) there is more than a single goal state.

Theorem 2. *Given a* δ_{edge} *that removes the same edge from* G_{det} *and* G_{prob}*, there exists a starting state, time* t*,* $T_{\max}$*, and performance threshold* θ*, for which both* $X_{\text{prob}} \succeq^{\lfloor \mathcal{F}(\theta) \rfloor} X_{\text{det}}$ *and* $X_{\text{prob}} \succeq^{\lfloor \mathcal{P}(t) \rfloor} X_{\text{det}}$.

Proof. Removing an edge from G_{det} cuts off at least one state from reaching a goal, since every edge is on the shortest path from a state to the nearest goal. Let s' denote a state that is cut-off from a goal. The probability of success for s' is zero for all time, $\forall t \geq 0\ P_{G|\mathbb{S}_0}(\texttt{true}|s'; t, X_{\text{det}}) = 0$, which means that both average and worst case efficiency is zero (regardless of the cutoff time $T_{\max}$), and that average and worst case power is always zero for the deterministic algorithm. By contrast, multiple paths from any node to the goal exist for the probabilistic algorithm, because every other state is incident to s' in G_{prob}, yielding $\forall t \geq \max_s T^*(s)\ P_{G|\mathbb{S}_0}(\texttt{true}|s'; t, X_{\text{prob}}) > 0$. The max operator accounts for the neighbor of s' farthest from the goal. Consequently, average and worst case efficiency are positive for $\theta > 0$, and average and worst case power are also strictly greater than zero. Thus, there are values of $\theta > 0$ and $t < \infty$, such that $X_{\text{prob}} \succeq^{\lfloor \mathcal{F}(\theta) \rfloor} X_{\text{det}}$ and $X_{\text{prob}} \succeq^{\lfloor \mathcal{P}(t) \rfloor} X_{\text{det}}$. □

E. DISCUSSION

This section illustrated how adding assumptions about the structure of the world and algorithms can lead to theoretical properties about the relative resilience of two algorithms. Like the previous case study, this case study also suggests a tradeoff between (a) algorithms that are efficient in unperturbed worlds, and (b) less efficient algorithms that are more powerful in a perturbed world. A common property from the two case studies is that, when tolerance is low (e.g., θ is high), the ability to follow multiple paths may decrease efficiency for unperturbed problems, but increase power for perturbed problems. Evaluating tradeoffs in terms of multiple paths from states to goals suggest a possible area for future work,

namely using graph-based analyses of the behavior potential to analyze resilience. Work on percolation theory suggests that strong theoretical results are possible for understanding resilience for complicated behavior in graphs (e.g., [47]).

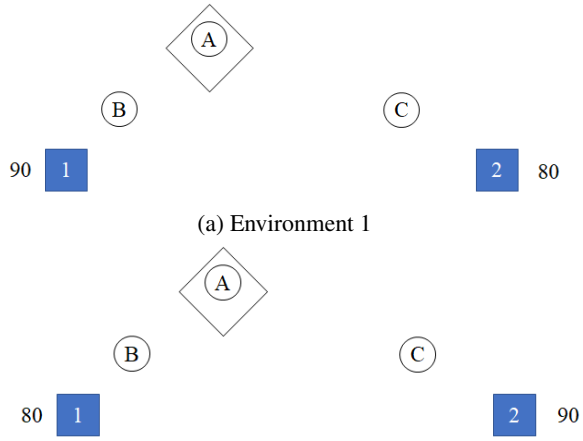
VI. CASE STUDY: EMPIRICALLY ESTIMATING RESILIENCE

The previous case studies evaluated resilience properties for systems that were small enough to enumerate trajectories, or structured enough to provide theoretical analysis. Each section showed a type of power-efficiency tradeoff, where algorithms that generated more trajectories had higher power, but lower efficiency than algorithms that generated optimal trajectories. This case study demonstrates that algorithms capable of exploring larger regions of the state space have higher power than algorithms that make decisions more quickly. However, the relative power of the different algorithms is nuanced. Importantly, this section demonstrates how sampling trajectories for a complicated distributed algorithm yields insights into the relative resilience of various algorithms. Sampling trajectories is done in the carefully constructed experiment. Power and efficiency are used to compare algorithm responses to various world and perturbation conditions.

This case study focuses on a best-of-N problem [48], [49], which requires a robotic collective to find a set of spatially distributed sites and form a consensus to select the highest value site. Fig. 3 illustrates two example environments containing a single hub, two sites with associated values, and three robots. The collective's behavior is inspired by the way biological collectives search for resources [48], [50]. Bio-inspired solutions to the problem have previously been shown to offer useful algorithms for disaster events over large spatial areas (e.g., earthquakes or wildfires), even when individual robots are subject to significant loss or perturbation [18].

A. ENVIRONMENT

The environment is a simulated two-dimensional world containing N sites $\{\sigma_1, \ldots, \sigma_N\}$, spatially distributed in the world and each assigned a quality. The environment includes M homogeneous point-based robots (e.g., no mass, no volume [51]) that know about a centralized hub location, can move about in the world by executing movement commands, can communicate with other robots within the hub, and can sense the quality of encountered sites. The collective behavior of the set of robots emerges from the individual robot activities through a distributed collective algorithm. Thus, the state of the world, which is referred to as the *world state* to distinguish it from the computational states of individual robots, includes the locations of the N sites, the site qualities, the hub location, the locations of the robots, and the internal computational states of the individual robots. The *collective actions* in the world are the robot movements and their sensing and communication acts. A robot can only communicate with other robots if both are at the hub.



(a) Environment 1

(b) Environment 2

FIGURE 3: Two best-of-N problem environments that illustrate how the environment affects interaction timing. The diamond represents a collective's hub. The blue squares represent sites, each with a quality value. The circles represent robots. Robot *A* will be recruited by Robot *B* to the closer site (Site 1) in both cases. (a) The highest value site is closer to the hub. (b) The highest value site is further from the hub.

B. COLLECTIVE GOAL

The collective's *goal* for the best-of-N problem is to discover and select the highest quality site in the environment. Robots at the hub regularly interact to determine if other robots favor a higher quality site option. Once a quorum of robots favors a particular site, they will commit to it [52], [53]. Robots will transition from consensus formation to decision execution when they detect a sufficient number of neighboring robots are also committed to the same site.

C. ALGORITHMS

Two collective algorithms are considered. Both algorithms rely on emergent collective behavior from individual robots, where each robot uses a *finite state machine* (FSM). The computational states in the FSM are referred to as *robot states* to differentiate them from the *world state*. Each robot i can only observe the environment within a limited sensing range, which means that what is observable by robot i, S^i_{obs}, is a small subset of the world state. The computational state of robot i, S^i_{comp}, stores current sensor values, task state, navigation information, which neighboring robots are communicating, and the location and quality of its favored site. Robot i's actions, A^i_{eff}, are movements and sensing in the world, and communications to nearby robots (if at the hub). Inter-robot communication includes whether a robot is in a favoring or committed state, votes received for quorum sensing, and the locations/qualities of sites known to other robots. Robots do not know the identity of other robots. Site locations and values are not known when the task begins.

The first algorithm, X_{core}, implements Reina et al.'s robot-based model for collective decision making [53]. The second

algorithm, X_{ext}, includes the extensions by Cody and Adams [54], [55]. The robots in X_{core} use Reina et al.'s defined probabilistic state transitions [53], [56]. The robots in X_{ext} use a probabilistic finite state machine developed by Cody and Adams [54], [55] to address the effects of *environmental bias* [57], [58] observed with X_{core} [49], [53], [56].

Individual robot *state transitions* include: *discovering* and favoring new site options, *recruiting* uncommitted robots, *abandoning* a previously favored site option, *inhibiting* other robots from favoring other site options, and *committing* to a site [52], [53]. Formally, robots probabilistically transition between five computational *states* in S_{comp}: uncommitted interactive, uncommitted latent, favoring interactive, favoring latent, and committed. Interactive and latent states differentiate whether robots are receptive to interaction. Robots in an uncommitted or favoring state can be recruited by other robots, while robots in the committed state cannot be recruited. Robots move from a favoring state to a committed state due to interactions with other robots; therefore, a robot's state transitions depend on interaction timings and contents.

D. TRAJECTORIES AND BEHAVIOR POTENTIAL

Each unique sequence of robot state transitions, inter-robot interactions, robot sensor readings, and robot movements generates a state-action *trajectory* according to Equation (1). The *behavior potential* of each algorithm is the union of all possible state-action trajectories. The trajectory is sensitive to timing effects, as illustrated in the following two examples.

The first example is a robot at the hub (robot A) interacting with other robots (B and C) returning to the hub from two different sites (1 and 2), as shown in Fig. 3. The two possible environments that result in different intermediate collective states. Site 1 (value = 90), which is closest to the hub in Environment 1 (Fig. 3a), has a higher value than Site 2 (value = 80). Robot B discovers Site 1, returns to the hub before robot C, and succeeds in recruiting robot A to investigate and favor Site 1. Robot C will not subsequently succeed in recruiting robot A, because robot C's favored site has a lower value than robot A's preferred site. Two robots will favor the higher valued Site 1, and only one robot will favor Site 2. The collective will be more likely to select Site 1, because robots can be recruited for the closer site at a faster rate.

The second example, shown in Environment 2 (Fig. 3b), changes the sites' relative values, as Site 1 has a lower value (80) than Site 2 (value = 90). Robot B again discovers Site 1, returns to the hub before robot C, and successfully recruits robot A, who subsequently leaves the hub to visit Site 1 to verify the site's value. Robot C will be unable to recruit robot A; thus, two robots will favor the lower valued site, resulting in a different intermediate collective state than in Environment 1. However, upon robot A's return, it can be recruited by robot C to investigate Site 2. The collective is more likely to select inferior Site 1.

E. EXPERIMENT DESIGN

A simulation-based experiment used the power metric to analyze and understand some resilience properties of the two collective algorithms. The experiment introduced an *ablation perturbation* that disabled a percentage of the collective's robots partway through each trial. The disabled robots were randomly selected from the collective's population, regardless of the robots' states or locations in the environment.

The hypothesis is that X_{core} and X_{ext} will exhibit similar performance levels, relative to their respective baselines, for perturbations affecting the collective's size. Each trial commenced with all robots within the 3 meters (m) x 3m hub region in the uncommitted interactive state. Robots with a 12.8m perception range searched a region up to 500m from the hub at a speed of 2.78m per timestep. Robots' sensory regions enclosed a full 360° circle around the robot. Each trial completed after a single decision. All trials completed after at most 25,000 timesteps. A single perturbation event occurred during each trial. The independent variables, listed in Table 1, included the number of robots in the collective at the start of the trial, the four site locations and qualities, and the perturbation timing relative to the collective's progress towards a decision. Each site had a unique quality value. Baseline results, in which no perturbation occurred, were also collected for each combination of independent variables.

TABLE 1: Independent variables.

Variable	Values
Number of Robots	50, 100, 200, 300, 400, 500
Site Value	60, 70, 80, 90
Site Locations	250-400m from the hub, at 3, 6, 9, and 12 on a clock
Perturbation Timing	15%, 35%, 60%
Robots Removed	25%, 50%

TABLE 2: Site configuration parameters.

ID	Site Value				Difficulty
	0	1	2	3	
SC.1	60	70	80	90	Easy
SC.2	70	80	90	60	Easy
SC.3	80	90	60	70	Intermediate
SC.4	90	60	70	80	Hard

The site configuration defined the problem difficulty. The site locations were constant across all trials, while the site values varied, as listed in Table 2. The distances for sites 0, 1, 2, and 3 were set to 400m, 350m, 300m, and 250m, respectively. Site 0 was located at the 9 o'clock position, site 1 at the 6 o'clock position, continuing in a counter-clockwise pattern. A problem was *easy* if the highest valued site was one of the two closest sites to the hub (i.e., at sites 2 or 3). The highest valued site was most distant from the hub (i.e., at site 0) for *hard* problems. Robots did not know any site locations at the start of the trial.

The collective's progress towards making a decision determined the perturbation's timing. Robots sensed a quorum

and committed to a site when at least 75% of the fifteen most recently received votes supported a single site. Since robot interactions were restricted to the hub where a robot has an equal probability of interacting with any other robot, the collective's decision progress can be approximated by a random robot's progress towards sensing a quorum [53]. Three perturbation timings were considered. Early perturbations occurred when 15% of robots favored the same site. Perturbations at 35% consensus occurred approximately halfway through the decision process. The late timing at 60% consensus evaluated the collective's ability to alter its decision quickly.

The dependent variables support estimating the average power and average efficiency metrics. *Selection accuracy* was measured by the percentage of trials in which the collective selected the best available site when the decision was made. *Decision time* was measured by the number of simulation iterations required to reach a quorum. Selection accuracy equals $P_{G|\mathbb{S}_0}$, and decision time equals t_{sat}.

F. RESULTS

The baseline and perturbation power and efficiency results are presented. Results are reported using the difference tests in Equation (6) for efficiency, and Equation (9) for power. The perturbation results are aggregated by perturbation timing and by the percentage of robots removed during the trial.

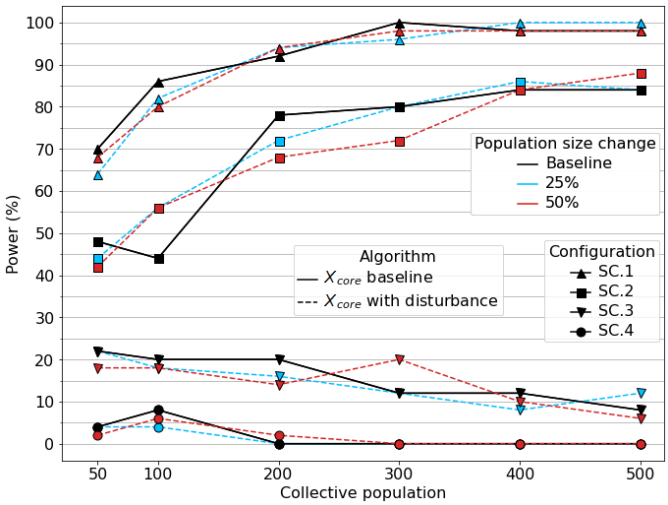
1) Power Metric

The power metric represents the success probability achieved as a function of time. Power was evaluated at $t = 7200$ iterations (i.e., two hours, 1 iteration = 1 second), which was subjectively chosen by how well it allowed the algorithms to be compared. The power metric illuminates how the algorithms' success probability is affected by the various independent variables and the perturbation.

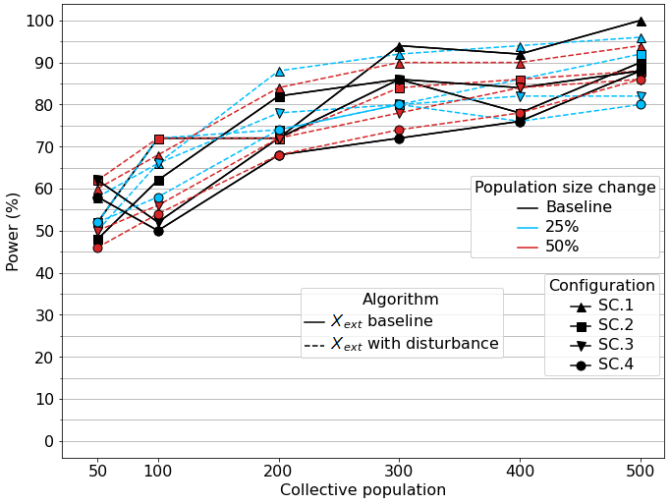
Both algorithms demonstrated a general increase in power as the collective size increased, which was an anticipated trend [59]. X_{core}'s power was clearly differentiated by problem difficulty. The power for the two easy configurations (SC.1 and SC.2) ranged between 64-100% and 42-88%, respectively, shown in Fig. 4a. The intermediate (SC.3) problem's power never exceeded 25%, and the hard (SC.4) problem's power ranged from 0-8% for collectives with less than 300 robots, and was zero for larger collective sizes.

X_{ext}'s ability to mitigate environmental bias is demonstrated by the higher power levels for the intermediate and hard problems, shown in Fig. 4b, compared to X_{core}. X_{ext}'s power was relatively consistent across the problem difficulties. The two easy configurations had power ranging from 52-62% with 50 robots to 88-96% with 500 robots. The intermediate problem ranged from 50-86%, and the hard problem had a power from 46-86%.

Comparing the resilience power between the algorithms demonstrates how well each algorithm performs under various conditions, as shown in Table 3. The difference test in Equation (9) was used: positive values in these tables indicate X_{ext} had a higher power, and negative values indicate



(a) X_{core}



(b) X_{ext}

FIGURE 4: Resilience power for the Remove Robots perturbation at decision time $t = 7200$ iterations.

X_{core}'s power was higher. X_{core} outperforms X_{ext} in the SC.1 configuration both in the baseline case and when the perturbation occurred. Notably, X_{core}'s higher performance occurred regardless of the perturbation's timing, or the percentage of robots removed, shown in the SC.1 rows in Tables 4 and 5, respectively. X_{ext} performed slightly better than X_{core} in the SC.2 configuration, and markedly better for the two harder configurations (SC.3 and SC.4).

2) Efficiency Metric

The efficiency metric represents the minimum time needed to achieve a given success probability. The success probability threshold was subjectively chosen as $\theta = 50\%$. Relative efficiency was computed using the difference in Equation (6). Only $t_{\text{sat}}(X_{\text{ext}})$ is reported, rather than the difference in times when X_{core} did not reach the success threshold. X_{ext} was more efficient than X_{core} in nearly all cases, shown in Tables 6–8.

TABLE 3: Relative power between the two algorithms for the baseline case given the configuration (Config.). Values > 0 indicate X_{ext} had a higher power than X_{core}.

Config.	Number of Robots					
	50	100	200	300	400	500
SC.1	-18	-14	-20	-6	-6	2
SC.2	0	18	4	6	-6	6
SC.3	40	32	52	74	72	80
SC.4	54	42	68	72	76	88

TABLE 4: Relative power between the algorithms after the perturbation occurred, by perturbation timing (% consensus). Values > 0 indicate X_{ext} had a higher power than X_{core}.

Config.	Timing	Number of Robots					
		50	100	200	300	400	500
SC.1	15	-15	-26	-17	-8	-2	-6
SC.1	35	-7	-4	-7	-8	-12	-1
SC.1	60	-12	-14	-8	-12	-5	-2
SC.2	15	15	20	0	8	6	2
SC.2	35	13	9	-3	0	-1	4
SC.2	60	7	12	7	5	3	8
SC.3	15	25	43	59	62	74	85
SC.3	35	31	36	61	59	76	71
SC.3	60	32	40	60	64	65	72
SC.4	15	46	52	61	78	74	83
SC.4	35	45	52	70	72	77	79
SC.4	60	46	56	72	75	76	84

TABLE 5: Relative power between the algorithms after the perturbation occurred, by the percentage of robots removed. Values > 0 indicate X_{ext} had a higher power than X_{core}.

Config.	% Removed	Number of Robots					
		50	100	200	300	400	500
SC.1	25	-6	-16	-6	-4	-6	-4
SC.1	50	-8	-12	-10	-8	-8	-4
SC.2	25	8	16	2	0	0	8
SC.2	50	20	16	4	12	2	0
SC.3	25	28	48	62	68	74	70
SC.3	50	32	38	58	58	74	80
SC.4	25	48	54	74	80	76	80
SC.4	50	44	48	66	74	78	86

X_{core} did not achieve the 50% success probability threshold in the SC.3 or SC.4 configurations, regardless of the perturbation timing or the percentage of robots removed.

The algorithms demonstrated similar levels of relative efficiency independent of the perturbation's timing. For example, with 200 or more robots in the baseline SC.1 configuration the minimum improvement in efficiency for X_{ext} over X_{core} ranged from a minimum of 1285 iterations (300 robots), to a maximum of 1590 iterations (500 robots), shown in Table 6. The post-perturbation efficiency, when considering the perturbation's timing, showed X_{ext} to be 1313 (200 robots, 60% consensus) to 1575 (500 robots, 60% consensus) iterations faster than X_{core} for the SC.1 configuration, shown in Table 7. This result indicates that momentum towards a decision was established relatively early in the decision making process. Removing robots after momentum for a site has been established does not alter the collective's eventual decision.

TABLE 6: Relative efficiency (in iterations) between the algorithms for the baseline case. Values > 0 indicate X_{ext} had a higher efficiency than X_{core}. (bold: X_{core} did not achieve 50% success probability, N/A: neither algorithm achieved 50% success probability.)

Config.	Number of Robots					
	50	100	200	300	400	500
SC.1	-16	1406	1483	1285	1460	1590
SC.2	N/A	**3977**	2161	2450	2161	2640
SC.3	**4320**	**5714**	**4250**	**3473**	**3435**	**3301**
SC.4	**4277**	**5946**	**4024**	**4036**	**3572**	**3599**

TABLE 7: Relative efficiency between the algorithms after the perturbation, by perturbation timing (% consensus). Values > 0 indicate X_{ext} had a higher efficiency than X_{core}. (bold: X_{core} did not achieve 50% success probability, N/A: neither algorithm achieved 50% success probability.)

Config.	Timing	Number of Robots					
		50	100	200	300	400	500
SC.1	15	1188	480	1419	1398	1330	1330
SC.1	35	1511	1135	1466	1404	1398	1384
SC.1	60	972	1307	1313	1425	1482	1575
SC.2	15	**5543**	3205	2198	2378	2402	2411
SC.2	35	2666	2590	2552	2472	2478	2553
SC.2	60	N/A	3312	2675	2624	2799	2457
SC.3	15	N/A	**4194**	**3700**	**3573**	**3615**	**3404**
SC.3	35	N/A	**4867**	**3957**	**3735**	**3604**	**3737**
SC.3	60	**5834**	**3774**	**3993**	**3473**	**3794**	**3574**
SC.4	15	**6195**	**5095**	**4097**	**3928**	**3908**	**3662**
SC.4	35	N/A	**4773**	**3973**	**4098**	**3711**	**3944**
SC.4	60	**6044**	**4784**	**3929**	**3768**	**3742**	**3574**

TABLE 8: Relative efficiency between the algorithms after the perturbation, by the percentage of robots removed. Values > 0 indicate X_{ext} had a higher efficiency than X_{core}. (bold: X_{core} did not achieve 50% success probability, N/A: neither algorithm achieved 50% success probability.)

Config.	% Removed	Number of Robots					
		50	100	200	300	400	500
SC.1	25	1283	985	1429	1438	1337	1498
SC.1	50	1204	1100	1426	1388	1460	1385
SC.2	25	**5686**	3062	2378	2345	2414	2448
SC.2	50	**5351**	2782	2536	2550	2645	2519
SC.3	25	N/A	**4088**	**3618**	**3506**	**3560**	**3641**
SC.3	50	N/A	**4641**	**4059**	**3776**	**3736**	**3431**
SC.4	25	**6100**	**5032**	**3797**	**3769**	**3800**	**3679**
SC.4	50	N/A	**4773**	**4258**	**4067**	**3795**	**3787**

The efficiency differences between the algorithms were relatively constant for collectives that began with 200 or more robots, regardless of the collective's size at decision time. For example, X_{ext} was 2161 iterations more efficient than X_{core} for the SC.2 configuration with 200 robots, and 2640 itera-

tions more efficient with 500 robots (Table 6). X_{ext} remained more efficient in the SC.2 configuration after the perturbation for an original collective size of 200 robots (Table 8), with an efficiency of 2378 iterations above X_{core} when removing 25% of the collective's robots, and 2536 iterations when removing 50%. X_{ext} was faster for collectives with an original size of 500 robots by 2448 iterations when removing 25% of the collective's robots, and 2519 iterations faster when removing 50%. The similar results by collective sizes indicates that the 300-500 sizes provided minimal improvement to the collective's decision time.

G. DISCUSSION

The power and efficiency metrics provide two important insights into the algorithms' performance. The relative power (Table 3) and efficiency (Table 6) metrics for the baseline case demonstrate that X_{ext} can accurately select the highest valued site in a wider variety of environmental configurations, and is faster to reach the correct decision than X_{core}. Comparing the baseline results to the post-perturbation results (e.g., Tables 3–8) demonstrates that both algorithms are *resilient* to the Remove Robots perturbation. The power and efficiency for each algorithm follow similar trends, regardless of the percentage of robots removed during runtime. However, these results also emphasize a characteristic of resilience that is consistent with the other case studies: *a resilient algorithm is not necessarily a high-performing algorithm*. An algorithm's actual success rate is as important as its performance in the presence of perturbations. Indeed, X_{ext} was designed to overcome the environmental bias that appeared in X_{core} by delaying transitions to favoring and commitment robot states, thereby allowing more trajectories to be explored [51], [54].

Two insights follow. First, if the performance threshold θ is increased, then X_{core} can have very low efficiency in the difficult worlds. The relevant data is not shown in the tables, but is easy to understand. X_{core} exhibits environmental bias, favoring nearby sites over distant sites. Consequently, X_{core} can choose an inferior site closer to the hub, rather than the best site farther from the hub, which means that it does not accomplish the goal as often as X_{ext} in the difficult world conditions. Efficiency depends on the performance tolerance θ, and for high values of θ X_{core} does not meet the performance criterion and is less efficient than X_{ext} in the difficult worlds. Second, the reason that X_{ext} is more powerful is that it favors different trajectories than X_{core}. Subjective observations confirm that X_{ext} causes robots that discover nearby sites of low quality to wait at the hub until other robots return instead of immediately transitioning to a recruiting state. This delay creates trajectories where distant high quality sites are reported to the hub, allowing recruiting to the highest quality site. X_{core} produces those trajectories far less often. X_{ext} permits more trajectories to be explored before making a decision, resulting in higher power in difficult worlds.

The conclusion is that X_{ext} is the superior algorithm for the information gathering task, as it demonstrates greater power and efficiency than X_{core}. This conclusion is based on two assumptions: the only disruption to the collective's behavior is the loss of robots during runtime, and that the problem difficulty varies. X_{ext} may not be resilient to other types of perturbations, or the possible operating environments may only pose easy decision problems. Declaring an algorithm or system to be resilient is specific to the type of perturbation encountered during the evaluation [38], [60].

VII. CONCLUSION AND FUTURE WORK

Resilience has mostly been studied and defined for agents with maintenance goals, roughly as an agent's ability to maintain some equilibrium state. However, resilience for agents with achievement goals may require the agent to change its entire operation. This manuscript proposed a novel definition for resilience for such agents that need to cope with unexpected perturbations to their task environment in order to achieve their goals. A theoretical framework was introduced that formally defines core notions related to resilience in agents with achievement goals, and also provides several metrics to quantify an agent's resilience. Critically, the framework's definitions and metrics provide a common mechanism for describing and comparing algorithm performance for embodied agents, regardless of the task or goal type. The metrics in the existing literature only applied to certain types of agents or goals. Three case studies demonstrate how the framework can be applied in real-world conditions to experimentally validate one's hypotheses about an agent's resilience: through complete analysis of all trajectories in bounded-memory algorithms, through theoretical analysis of highly structured problems, and through sampling-based estimations of complex distributed algorithms. The provided framework and metrics can also inform the design of resilient agents with achievement goals.

Designing resilient systems is challenging due to the computational complexity inherent in evaluating all possible trajectories of a robotic system operating in a real-world environment. An exact measurement of resilience in embodied agents will require designers to consider every possible environmental condition or perturbation to an agent's behavior potential, and the subsequently changed behavior potentials, which is practically only possible for small well-defined domains. Fortunately, determining absolute resilience is often less important than estimating relative resilience, because comparing different algorithms' performance is often more useful than deriving a raw performance measurement. The three case studies; thus, suggest a tradeoff between choosing optimal behaviors under unperturbed conditions and having the ability to exploit multiple paths to achieve a goal under perturbed conditions. This tradeoff manifested itself in the case studies as a tradeoff between the efficiency and power metrics, which designers can evaluate.

Future work will further explore how estimating competency and resilience can mitigate the challenges in computing these metrics exactly. Design patterns that enable adaptive and resilient behavior in goal-based agents will be identified.

REFERENCES

[1] E. Krotkov, D. Hackett, L. Jackel, M. Perschbacher, J. Pippine, J. Strauss, G. Pratt, and C. Orlowski, "The DARPA robotics challenge finals: Results and perspectives," *Journal of Field Robotics*, vol. 34, no. 2, pp. 229–240, 2017.

[2] G.-Z. Yang, J. Bellingham, P. E. Dupont, P. Fischer, L. Floridi, R. Full, N. Jacobstein, V. Kumar, M. McNutt, R. Merrifield, B. J. Nelson, B. Scassellati, M. Taddeo, R. Taylor, M. Veloso, Z. L. Wang, and R. Wood, "The grand challenges of Science Robotics," *Science Robotics*, vol. 3, no. 14, p. eaar7650, 2018.

[3] S. L. Pimm, "The complexity and stability of ecosystems," *Nature*, vol. 307, no. 5949, pp. 321–326, 1984.

[4] V. Grimm and C. Wissel, "Babel, or the ecological stability discussions: An inventory and analysis of terminology and a guide for avoiding confusion," *Oecologia*, vol. 109, no. 3, pp. 323–334, 1997.

[5] F. S. Brand and K. Jax, "Focusing the Meaning(s) of Resilience: Resilience as a Descriptive Concept and a Boundary Object," *Ecology and Society*, vol. 12, no. 1, 2007.

[6] P. Martin-Breen and J. M. Anderies, "Resilience: A Literature Review," Institute of Development Studies (IDS), Brighton, Tech. Rep., 2011.

[7] S. Hosseini, K. Barker, and J. E. Ramirez-Marquez, "A review of definitions and measures of system resilience," *Reliability Engineering and System Safety*, vol. 145, pp. 47–61, 2016.

[8] R. Bhamra, S. Dani, and K. Burnard, "Resilience: the concept, a literature review and future directions," *International Journal of Production Research*, vol. 49, no. 18, pp. 5375–5393, 2011.

[9] L. Olsson, A. Jerneck, H. Thoren, J. Persson, and D. O'Byrne, "Why resilience is unappealing to social science: Theoretical and empirical investigations of the scientific use of resilience," *Science Advances*, vol. 1, no. 4, p. e1400217, 2015.

[10] A. E. Quinlan, M. Berbés-Blázquez, L. J. Haider, and G. D. Peterson, "Measuring and assessing resilience: Broadening understanding through multiple disciplinary perspectives," *Journal of Applied Ecology*, vol. 53, no. 3, pp. 677–687, 2016.

[11] R. Francis and B. Bekera, "A metric and frameworks for resilience analysis of engineered and infrastructure systems," *Reliability Engineering and System Safety*, vol. 121, pp. 90–103, 2014.

[12] R. Patriarca, J. Bergström, G. Di Gravio, and F. Costantino, "Resilience engineering: Current status of the research and future challenges," *Safety Science*, vol. 102, pp. 79–100, 2018.

[13] C. S. Holling, "Engineering Resilience versus Ecological Resilience," in *Engineering Within Ecological Constraints*. Washington, D.C.: The National Academies Press, 1996, pp. 31–43.

[14] G. S. Cumming and G. D. Peterson, "Unifying Research on Social-Ecological Resilience and Collapse," *Trends in Ecology & Evolution*, vol. 32, no. 9, pp. 695–713, 2017.

[15] M. B. Van Riemsdijk, M. Dastani, and M. Winikoff, "Goals in agent systems: A unifying framework," in *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems*, vol. 2, 2008, pp. 702–709.

[16] M. A. Goodrich, J. A. Adams, and M. Scheutz, "Autonomy reconsidered: Towards developing multi-agent systems," in *Proceedings of SAI Intelligent Systems Conference*. Springer, 2021, pp. 573–592.

[17] M. Egerstedt, J. N. Pauli, G. Notomista, and S. Hutchinson, "Robot ecology: Constraint-based control design for long duration autonomy," *Annual Reviews in Control*, vol. 46, pp. 1–7, 2018.

[18] J. Leaf and J. A. Adams, "Measuring Resilience in Collective Robotic Algorithms," in *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, 2022, pp. 1666–1668.

[19] X. Cao, P. Jain, and M. A. Goodrich, "Adapted metrics for measuring competency and resilience for autonomous robot systems in discrete time markov chains," in *IEEE International Conference on Systems, Man, and Cybernetics*. IEEE, 2022, pp. 71–76.

[20] P. Albertos and I. Mareels, *Feedback and Control for Everyone*. Berlin: Springer-Verlag, 2010.

[21] A. M. Madni and S. Jackson, "Towards a conceptual framework for resilience engineering," *IEEE Systems Journal*, vol. 3, no. 2, pp. 181–191, 2009.

[22] R. C. Lewontin, "The meaning of stability," in *Brookhaven Symposia in Biology*, vol. 22, 1969, pp. 13–24.

[23] C. S. Holling, "Resilience and Stability of Ecological Systems," *Annual Review of Ecology and Systematics*, vol. 4, pp. 1–23, 1973.

[24] T. G. Lewis, "The many faces of resilience," *Communications of the ACM*, vol. 66, no. 1, pp. 56–51, 2022.

[25] C. Béné and L. Doyen, "From Resistance to Transformation: A Generic Metric of Resilience Through Viability," *Earth's Future*, vol. 6, pp. 979–996, 2018.

[26] S. Martin, "The Cost of Restoration as a Way of Defining Resilience: a Viability Approach Applied to a Model of Lake Eutrophication," *Ecology and Society*, vol. 9, no. 2, 2004.

[27] C. Rougé, J.-D. Mathias, and G. Deffuant, "Extending the viability theory framework of resilience to uncertain dynamics, and application to lake eutrophication," *Ecological Indicators*, vol. 29, pp. 420–433, 2013.

[28] R. R. Hoffman and P. A. Hancock, "Measuring Resilience," *Human Factors*, vol. 59, no. 4, pp. 564–581, 2017.

[29] E. Seraj, L. Chen, and M. C. Gombolay, "A Hierarchical Coordination Framework for Joint Perception-Action Tasks in Composite Robot Teams," *IEEE Transactions on Robotics*, vol. 38, no. 1, pp. 139–158, 2022.

[30] H. T. Tran, J. C. Domerçant, and D. N. Mavris, "A Network-based Cost Comparison of Resilient and Robust System-of-Systems," *Procedia Computer Science*, vol. 95, pp. 126–133, 2016.

[31] K. Saulnier, D. Saldana, A. Prorok, G. J. Pappas, and V. Kumar, "Resilient Flocking for Mobile Robot Teams," *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 1039–1046, 2017.

[32] G. Bai, Y. Li, Y. Fang, Y. A. Zhang, and J. Tao, "Network approach for resilience evaluation of a UAV swarm by considering communication limits," *Reliability Engineering and System Safety*, vol. 193, no. June 2019, p. 106602, 2020.

[33] T. G. Lewis, "The mathematics of catastrophe," *AppliedMath*, vol. 2, no. 3, pp. 480–500, 2022.

[34] M. Bruneau, S. E. Chang, R. T. Eguchi, G. C. Lee, T. D. O'Rourke, A. M. Reinhorn, M. Shinozuka, K. Tierney, W. A. Wallace, and D. Von Winterfeldt, "A Framework to Quantitatively Assess and Enhance the Seismic Resilience of Communities," *Earthquake Spectra*, vol. 19, no. 4, pp. 733–752, 2003.

[35] C. W. Zobel, "Representing perceived tradeoffs in defining disaster resilience," *Decision Support Systems*, vol. 50, pp. 394–403, 2010.

[36] D. Henry and J. Emmanuel Ramirez-Marquez, "Generic metrics and quantitative approaches for system resilience as a function of time," *Reliability Engineering & System Safety*, vol. 99, pp. 114–122, 2012.

[37] J. Ingrisch and M. Bahn, "Towards a Comparable Quantification of Resilience," *Trends in Ecology & Evolution*, vol. 33, no. 4, pp. 251–259, 2018.

[38] D. Hodgson, J. L. McDonald, and D. J. Hosken, "What do you mean, 'resilient'?" *Trends in Ecology & Evolution*, vol. 30, no. 9, pp. 503–506, 2015.

[39] B. Walker, C. S. Holling, S. R. Carpenter, and A. Kinzig, "Resilience, Adaptability and Transformability in Social-ecological Systems," *Ecology and Society*, vol. 9, no. 2, 2004.

[40] L. H. Gunderson and C. S. Holling, Eds., *Panarchy: Understanding transformations in human and natural systems*. Washington, D.C.: Island Press, 2001.

[41] D. G. Angeler and C. R. Allen, "Quantifying resilience," *Journal of Applied Ecology*, vol. 53, pp. 617–624, 2016.

[42] B. M. Spears, S. C. Ives, D. G. Angeler, C. R. Allen, S. Birk, L. Carvalho, S. Cavers, F. Daunt, R. D. Morton, M. J. O. Pocock, G. Rhodes, and S. J. Thackeray, "Effective management of ecological resilience - Are we there yet?" *Journal of Applied Ecology*, vol. 52, no. 5, pp. 1311–1315, 2015.

[43] K. L. Nash, C. R. Allen, D. G. Angeler, C. Barichievy, T. Eason, A. S. Garmestani, N. A. J. Graham, D. Granholm, M. Knutson, R. J. Nelson, M. Nyström, C. A. Stow, and S. M. Sundstrom, "Discontinuities, cross-scale patterns, and the organization of ecosystems," *Ecology*, vol. 95, no. 3, pp. 654–667, 2014.

[44] R. J. Standish, R. J. Hobbs, M. M. Mayfield, B. T. Bestelmeyer, K. N. Suding, L. L. Battaglia, V. Eviner, C. V. Hawkes, V. M. Temperton, V. A. Cramer, J. A. Harris, J. L. Funk, and P. A. Thomas, "Resilience in ecology: Abstraction, distraction, or where the action is?" *Biological Conservation*, vol. 177, pp. 43–51, 2014.

[45] D. L. Baho, C. R. Allen, A. Garmestani, H. Fried-Petersen, S. E. Renes, L. Gunderson, and D. G. Angeler, "A quantitative framework for assessing ecological resilience," *Ecology and Society*, vol. 22, no. 3, 2017.

[46] S. T. Freedman and J. A. Adams, "The inherent components of unmanned vehicle situation awareness," in *IEEE International Conference on Systems, Man and Cybernetics*, 2007, pp. 973–977.

[47] M. Newman, *Networks*. Oxford University Press, 2018.

[48] T. D. Seeley, *The wisdom of the hive: The social physiology of honey bee colonies*. Cambridge: Harvard University Press, 1995.

[49] G. Valentini, E. Ferrante, and M. Dorigo, "The Best-of-n Problem in Robot Swarms: Formalization, State of the Art, and Novel Perspectives," *Frontiers in Robotics and AI*, vol. 4, p. 9, 2017.
[50] J. L. Deneubourg, S. Aron, S. Goss, J. M. Pasteels, and G. Duerinck, "Random behaviour, amplification processes and number of participants: How they contribute to the foraging properties of ants," *Physica D: Nonlinear Phenomena*, vol. 22, no. 1-3, pp. 176–186, 1986.
[51] J. R. Cody and J. A. Adams, "An evaluation of quorum sensing mechanisms in collective value-sensitive site selection," in *International Symposium on Multi-Robot and Multi-Agent Systems*, 2017, pp. 40–47.
[52] C. A. Parker and H. Zhang, "Cooperative decision-making in decentralized multiple-robot systems: The best-of-N problem," *IEEE/ASME Transactions on Mechatronics*, vol. 14, no. 2, pp. 240–251, 2009.
[53] A. Reina, G. Valentini, C. Fernández-Oto, M. Dorigo, and V. Trianni, "A Design Pattern for Decentralised Decision Making," *PLOS ONE*, vol. 10, no. 10, p. e0140950, 2015.
[54] J. R. Cody, "Discrete Consensus Decisions in Human-Collective Teams," Ph.D. dissertation, Vanderbilt University, 2018.
[55] J. R. Cody, K. A. Roundtree, and J. A. Adams, "Human-collective collaborative target selection," *ACM Transactions on Human-Robot Interaction*, vol. 10, no. 2, pp. 1–29, 2021.
[56] A. Reina, R. Miletitch, M. Dorigo, and V. Trianni, "A quantitative micro-macro link for collective decisions: the shortest path discovery/selection example," *Swarm Intelligence*, vol. 9, no. 2-3, pp. 75–102, 2015.
[57] T. Laomettachit, T. Termsaithong, A. Sae-Tang, and O. Duangphakdee, "Decision-making in honeybee swarms based on quality and distance information of candidate nest sites," *Journal of Theoretical Biology*, vol. 364, pp. 21–30, 2015.
[58] T. M. Schaerf, J. C. Makinson, M. R. Myerscough, and M. Beekman, "Do small swarms have an advantage when house hunting? The effect of swarm size on nest-site selection by Apis mellifera," *Journal of the Royal Society Interface*, vol. 10, p. 20130533, 2013.
[59] L. Bayındır, "A review of swarm robotics tasks," *Neurocomputing*, vol. 172, pp. 292–321, 2016.
[60] S. Carpenter, B. Walker, J. M. Anderies, and N. Abel, "From Metaphor to Measurement: Resilience of What to What?" *Ecosystems*, vol. 4, no. 8, pp. 765–781, 2001.

JENNIFER LEAF (GS) received the B.S. degree in computer science from Pacific Lutheran University, Tacoma, WA, USA, in 2001, the M.S. degree in Computing and Software Systems from the University of Washington, Tacoma, WA, USA, in 2007, and the Ph.D. degree in Robotics at Oregon State University, Corvallis, OR, USA in 2023. She worked as a Senior Computer Scientist at Advanced Systems Technology (2001-2006), NewTec (2006-2009), and ManTech (2010), and as a Senior Program Manager at Microsoft (2010-2015). She is an Assistant Professor in the Department of Mechanical Engineering and Technology at Eastern Washington University.

JULIE A. ADAMS (SM'01) received the Bachelor of Science degree in computer science in 1989 and the Bachelor of Business Administration degree in accounting in 1990 from Siena College, Albany, NY, USA, and the M.S.E. degree in 1993 and Ph.D. degree in 1995 in computer and information sciences from the University of Pennsylvania, Philadelphia, PA, USA. She is the Associate Director of Research for the Collaborative Robotics and Intelligent Systems Institute and a Professor in the School of Electrical Engineering and Computer Science, Oregon State University, where she directs the Human-Machine Teaming Laboratory. Her research focuses on distributed artificially intelligent algorithms and the development of complex human-machine systems for large human and robotic teams.

MATTHIAS SCHEUTZ is a Professor in Cognitive and Computer Science in the Department of Computer Science, Director of the Human-Robot Interaction Laboratory and the new Human-Robot Interaction Ph.D. and M.S. programs, and Bernard M. Gordon Senior Faculty Fellow in the School of Engineering at Tufts University. He earned a Ph.D. in Philosophy from the University of Vienna in 1995 and a Joint Ph.D. in Cognitive Science and Computer Science from Indiana University Bloomington in 1999. He has more than 400 peer-reviewed publications in artificial intelligence, natural language processing, cognitive modeling, robotics, and human-robot interaction. His current research focuses on resilient interactive autonomous systems with natural language and machine learning capabilities.

MICHAEL A. GOODRICH (SM'14) received the B.S., M.S., and Ph.D. degrees in electrical and computer engineering from Brigham Young University, Provo, UT, USA, in 1992, 1995, and 1996, respectively. From 1996 to 1998, he was a Research Associate with Nissan Cambridge Research, Nissan Research and Development, Inc., Cambridge, MA, USA. Since 1998, he has been with the Department of Computer Science, Brigham Young University, where he is currently a Professor and director of the Human-Centered Machine Intelligence Laboratory. His current research interests include human-robot interaction, human interaction with bio-inspired swarms, decision theory, multiagent learning, and human-centered engineering.