Sean C. Rismiller
Department of Mechanical Engineering,
Carnegie Mellon University,
Pittsburgh, PA 15213
e-mail: srismill@andrew.cmu.edu

Jonathan Cagan
Department of Mechanical Engineering,
Carnegie Mellon University,
Pittsburgh, PA 15213
e-mail: cagan@cmu.edu

Christopher McComb
School of Engineering Design, Technology, and
Professional Programs,
The Pennsylvania State University,
University Park, PA 16802
e-mail: uum209@psu.edu

An Adversarial Agent-Based Design Method Using Stochastic Stackelberg Game Conditions

Products must often endure challenging conditions while fulfilling their intended functions. Game-theoretic methods can readily create a wide variety of these conditions to consider when creating designs. This work introduces Cognitively Inspired Adversarial Agents (CIAAs) that use a Stackelberg game format to generate designs resistant to these conditions. These agents are used to generate designs while considering a multidimensional attack. Designs are produced under these adversarial conditions and compared to others generated without considering adversaries to confirm the agents' performance. The agents create designs able to withstand multiple combined conditions.
[DOI: 10.1115/1.4049862]

Keywords: agent based systems, robust design, Stackelberg game, agent-based design

1 Introduction: Cognitively Inspired Simulated Annealing Teams and Robust Design Via Stackelberg Games

McComb et al. [1] introduced the Cognitively-Inspired Simulated Annealing Teams (CISAT) framework, an agent-based simulated annealing (SA) framework developed to simulate and study human design teams. Agent-based systems such as these simulate individual agents, representing anything that can act on a given modeled environment, and their actions to create complex processes and results [2]. As an agent-based system, CISAT evolves a design from the actions and interactions of groups of agents, which each act by stochastically applying design configuration rules (grammars) to change their designs. These changes are then accepted or rejected according to a simulated annealing (SA) algorithm [3]. By using SA as the search strategy, agents are directed toward globally optimized designs and require only objective function information rather than gradient information, enabling them to solve a wide range of problems with this method. By using a variety of design grammars, agents can move through the design space in different ways to efficiently explore complex design spaces and address difficult objective functions with multiple local minima and discontinuities. Importantly, CISAT agents seek to determine preferred designs based on an objective goal, but they have not yet been applied to accounting for challenging conditions in design.

Robust design methodologies present one way to resist these conditions by minimizing of the impact of variance of parameters on a design's feasibility and performance [4,5], enhancing their consistency and safety through various approaches [6–12]. Robust design may also be modeled as a Stackelberg game, in which players take turns making actions, using observations of other player's actions to inform their decisions [13,14]. A player is either a "leader" who is the first to commit to a strategy or a "follower" who responds in turn. These games have been applied to various design problems [14–19]; with respect to design, the designer is the leader as they must commit to a design by deploying or producing it, after which they cannot adjust it "in the field" as they could before committing. Adversaries then attack that design with their given tools, making them followers. This process creates Stackelberg equilibria, where in addition to Nash equilibrium conditions, in which no player can benefit by deviating their strategy when considering other players' current strategies [20], the leader must also consider potential follower responses to any deviations it makes [21]. This method can readily challenge designs in a variety of ways beyond just the variance of parameters by using adversarial followers to model challenging conditions, creating optimal designs that are resistant to general worst-case attacks and scenarios.

Because of their ability to navigate multi-model spaces, CISAT agents are adaptable to solve design problems under a variety of complicated constraints. This work modifies these agents by integrating a Stackelberg game design process in them to create Cognitively Inspired Adversarial Agents (CIAAs). CIAAs incorporate adversary agents that compromise the design as it is developed in order to predict the possible responses to each action they take as designers. These new agents will enable ready development and study of solutions to open-ended engineering problems that can be attacked in multiple ways, such as a truss design which is used in this work. First, an explanation of the CIAAs is presented. Designs the CIAAs generate are then compared to designs created by CISAT (which do not consider attacks) to test the agents, examining mass, factor of safety, and ability for the design to remain feasible despite the adversarial attack.

2 Cognitively Inspired Adversarial Agents

2.1 System Description. CIAAs are computational engineering designer agents that use SA with the Triki adaptive annealing schedule [22]. The CIAA designer makes its changes while predicting potential responses and failures for the design, enabling it to take actions that account for these possibilities. These potential responses are generated by adversary agents assigned to the designer, who take turns changing and attacking the design, turning the process into a non-cooperative Stackelberg game. The use of adversary agents to generate and optimize these attacks enables adaptive search against the current design and narrows the list of possibilities to the most critical scenarios; however, the designer will only consider and design against scenarios that are within the adversaries' capabilities, which can be set to the conditions a design must endure. Multiple adversaries can be assigned to the designer to create a variety of independent attack plans that the designer must consider at once, and adversaries can make multiple moves after every designer move to better optimize these attacks.

The results of the adversaries guide the designer by influencing the score of its moves, leading it to probabilistically accept more

Contributed by the Design Automation Committee of ASME for publication in the Journal of Mechanical Design. Manuscript received September 12, 2020; final manuscript received January 18, 2021; published online January 29, 2021. Assoc. Editor: Jitesh H. Panchal.

moves that maintain resistance to the adversaries' attacks. For example, if the designer makes a move that resolves a vulnerability and protects the design against further attacks, the adversaries will perform much more poorly, giving the designer a corresponding improvement in its objective function and promoting that move. This structure is similar to the nested bilevel genetic algorithm in Liu et al. [17] which solves bilevel problems with discrete variables that are otherwise difficult to simplify. This work applies a similar structure to computational agents to solve a bilevel design and attack problem, enabling agents to consider future responses and change their actions to accommodate them.

Figure 1 illustrates the CIAA structure and shows adversary-controlled sections in shaded blocks. The CIAA designer first makes a change to the design. The adversaries then build their response to the designer's move with their given number of iterations to model the designer's prediction of how they'll respond. The designer next incorporates the results from the adversaries' responses into its decision to accept or reject its own iteration. In this way the designer and adversaries take turns optimizing their respective designs and attacks, iterating against each other and converging to a Stackelberg equilibrium, in which the adversaries have converged to the best attack plan for the given design, and the designer cannot improve on that design without introducing a vulnerability that the adversaries can then exploit. To ensure the designer and adversaries move at consistent rates relative to each other, they are permitted to try new moves if their current moves are rejected until their moves are accepted or a limit on attempts is reached. These modifications create expanded agents to conduct adversary-resistant design.

2.2 Designer and Adversary Grammars. Design grammars define the actions that the computational agents may take, and by extension how a design may be built and attacked. New possibilities may be introduced by adding new grammar rules to the agents. To create designs, the grammars therefore must include the elementary move operators that allow for basic progress and fine-tuning of solutions. It is also helpful to include higher-level rules that execute specific combinations of moves to achieve a certain result; this eliminates intermediate states that would prevent progress, as these states would be rejected for poor performance due to their incomplete nature. In this work, designer and adversary agents have differing move operators to accomplish their unique roles, but a designer and its assigned adversaries must both act on the same design representation that captures all changes made.

The designer move operators are based on those introduced by McComb, et al. [1], and encompass placement, removal, and adjustment of members and joints, forming the basis of truss design.

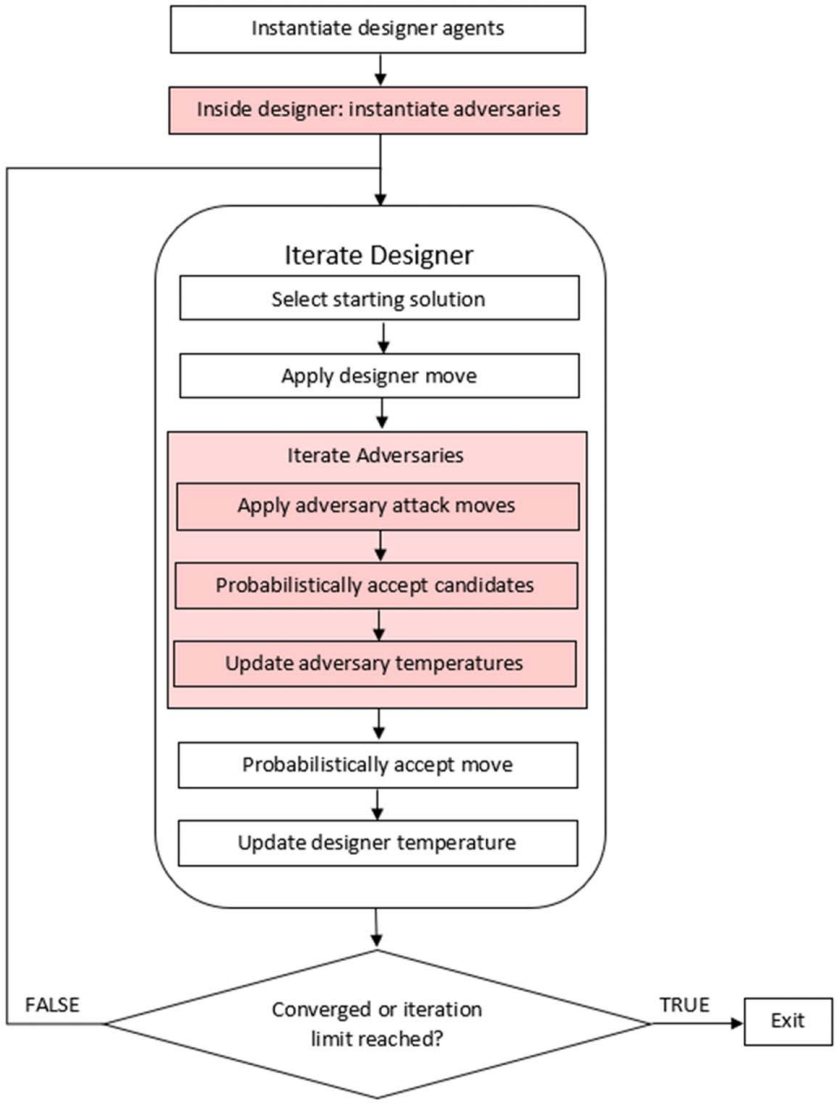

Fig. 1 Flowchart for a CIAA designer, adversary-controlled sections are in shaded boxes

Similarly, this work introduces move operators for adversaries to destroy members, load joints in any chosen direction, negate these actions, and change previously destroyed members and loaded joints, which combines negating an attack and selecting a new one of the same type. The "change" operators allow adversaries to adjust to more effective attacks without creating undesirable intermediate states in which they are not damaging the truss. Agents choose available moves stochastically and accept or reject them according to their results via SA. To set bounds on the scenarios the designer must consider, adversaries are limited in the number of members they may destroy, influenced by their thicknesses, and the number of loads they may apply. Changing these limits would allow for adjustment of the scenarios that the designer must consider. While member destruction and abnormal joint loading do not describe all the conditions a truss can face, their combination was chosen to challenge the designers with a wide variety of severe scenarios that they must account for.

2.3 Objective Functions. Objective functions guide the designer and adversaries to make moves that accomplish their goals by determining what moves are accepted or rejected, based on whether they improve the agents' objective functions or not. These functions therefore must reflect each agents' goals. In the truss design example, the designer's goal is to create a design with a low mass and high factor of safety that is also damaged as little as possible by the adversaries. Conversely, the adversaries each seek to damage the design as much as possible. For the truss problem, the objective functions for the designer and the adversaries, respectively, are

$$f_{designer} = m(x) - FOS(x) + g_{FOS}(x) + g_{invalid}(x) + D_{adv}(x, y) \quad (1)$$

$$f_{adversary} = -D_{adv}(x, y) \quad (2)$$

where x represents the current truss design when loaded normally, y represents the current adversary's attack plan, $m(x)$ is the mass of the design, $FOS(x)$ is the lowest factor of safety among all truss members in the designer's load case, $g_{FOS}(x)$ is a penalty function assigned if $FOS(x)$ falls below a given value (1.25 in this work), and $g_{invalid}(x)$ is a penalty assigned if the design is invalid, which occurs if it becomes unstable. These factors represent the designer's base goal and are defined in the work by McComb et al. [1]. $D_{adv}(y, x)$ represents the adversarial condition added in this work, namely an attack on the truss, and is defined as the damage the adversary does to the truss as follows:

$$D_{adv}(x, y) = m_d(x, y) + p_{invalid}(x, y) - g_{limit}(y) \quad (3)$$

where $m_d(x, y)$ is the sum of "damaged" mass, defined as the mass of truss members either directly destroyed, or yielded or buckled from the subsequent application of force by adversaries. $p_{invalid}(x, y)$ is a bonus given to an adversary if it renders the truss invalid with its attack, indicating collapse. $g_{limit}(y)$ is a penalty assigned if the adversary exceeds the maximum number of members they may destroy or the number of loads they may apply, enforcing the bounds on their abilities. The penalty functions fulfill the roles of constraints for SA by increasing the objective function to an unacceptable value if violated. $p_{invalid}(x, y)$, as a bonus to the adversary, also functions as a penalty to the designer if fulfilled, constraining the designer against making collapsible trusses. Objectives and penalties may be multiplied by constants to normalize their scales and generate solution choices for a given problem situation; however, relative comparison between solutions can be made within any chosen weighted objective set. This work explores a single set of objective function weightings to demonstrate the method. Varying weights, including those of the adversarial conditions against the designer's original goals, to generate a pareto surface of designs would allow for further exploration of the tradeoff between the base design goals and design resistance. Given these objective functions, the adversaries influence the designer by attacking vulnerabilities, driving up $D_{adv}(x, y)$ if they succeed and guiding the designer to accept moves that do not introduce those vulnerabilities, resulting in greater resistance. The designer influences the adversaries by controlling the space they work in, as they can only attack the design they are given.

2.4 Design Process. To conduct the design process, the designer is instantiated with a design from a set of seed designs, and within it the adversaries are instantiated with attacks on that design. The designer first applies a designer move operator (DMO) to modify the design. Before accepting or rejecting its DMO, the adversaries apply their adversary move operators (AMOs), optimizing their attacks to find and exploit vulnerabilities. The adversaries' AMOs are accepted or rejected based on the difference in f_{adv} immediately before and after each AMO per the SA algorithm. After the adversaries complete their AMOs, the designer then accepts or rejects its own DMO based on the difference between f_{design} from before its DMO and f^*_{design} after the adversaries move, where f^*_{design} represents the worst-case designer's objective function resulting from the adversaries' actions. If the designer's DMO is rejected, adversaries are also reverted to before the DMO was made. If the DMO is accepted, f^*_{design} becomes the new f_{design}. This has the effect of making the designer look ahead and predict how the adversaries will respond to its DMO before committing to it. Note that because adversaries evaluate their AMOs based on immediate changes in their objective functions, they do not look ahead in the same manner to see how the designer would respond to their actions. Despite this limitation, the presented method was effective in creating resistant designs; future work could explore removing these limitations and varying the depth of prediction to see how the agents' performance changes. The process ends once a maximum number of designer iterations have been completed. After completing all designer iterations, the designs are further tested against more thorough adversary attacks to confirm their effectiveness. In these additional attacks, the adversaries are given many more iterations to optimize, ensuring an extensive search for an effective attack. The fraction of these attacks that are unable to collapse the truss is taken as the metric for resistance.

3 Results

Figure 2 depicts the designer's problem statement with required joints. The two outer joints are supported with pinned connections. Adjacent to those are the loaded joints, with arrows indicating the direction of load. The center joint is supported with a roller connection. The loaded joints must be supported by a truss structure that connects them to the supports; this truss is optimized to fulfill its objective functions with the processes presented.

Truss designs produced under the adversarial conditions show significant differences from those produced using a non-adversarial process as in CISAT. All designs produced with the non-adversarial process collapsed in all adversary attacks during testing, indicating their lack of resistance to the conditions they must face. Designs produced using adversarial conditions with CIAAs were much

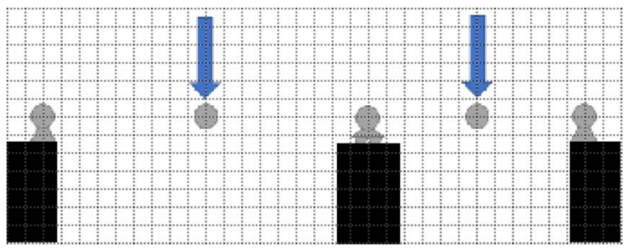

Fig. 2 Designer's truss design problem statement with supports and required loads

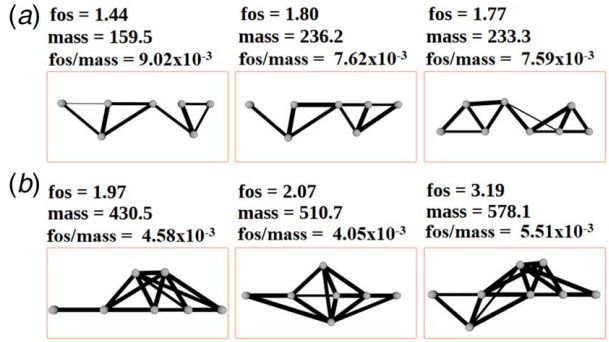

Fig. 3 Sample trusses produced under (*a*) non-adversarial conditions and (*b*) adversarial conditions

more capable of resisting these attacks without collapse, with some samples able to resist all attacks, confirming their ability to account for the adversaries. The fraction of adversary attacks survived is chosen as the indicator of resistance because there is insufficient information regarding degrees of damage when comparing CIAA-generated and CISAT-generated trusses. Because the CISAT-generated trusses collapsed in all attacks, there are no further degrees of damage to compare with CIAA-generated trusses; however, comparisons using this metric would be needed to compare different trusses that resisted attacks without collapse, and their respective design methods.

Figure 3 shows examples of CISAT-generated and CIAA-generated trusses. The CIAA-generated trusses shown survived all adversarial attacks. This resistance to attack incurs additional costs, indicated by the much greater mass of these trusses. While their FOSs are also typically greater, they are less efficient at sustaining the normally required load, indicated by the reduced ratios of FOS/mass which represents how efficiently the mass is being used. This reduced efficiency of the CIAA-generated designs is due to the redundant members that are required to stabilize it during adversary attacks, even though they provide less support during normal loading. This creates a tradeoff between resistance to attack and the original base objectives of maximizing FOS and minimizing mass.

Figure 4 demonstrates how trusses may resist attacks or collapse. In the CISAT-generated truss, a single-member destruction, indicated with a cross in Fig. 4(*a*), can render the structure unstable as shown in Fig. 4(*b*), with the disconnected section boxed. The lack of redundancy makes these trusses trivial to collapse, even if the adversary does not use all its tools. The CIAA-generated truss in Fig. 4(*c*) is subjected to a similar attack; however, it exhausts the adversary's ability to destroy members while remaining stable due to its redundant members. This prompts the adversary to load it in the direction of the arrow in Fig. 4(*d*); this loading makes the dotted member fail by buckling. The truss then stabilizes as shown in Fig. 4(*e*), as the remaining members pick up the load without failing, preventing the collapse of the truss. One possible load is shown, but adversaries can load any joint in any direction; their choices are optimized to maximize the damage they inflict or to collapse the truss if possible.

Due to the stochasticity of SA and the use of weighted objective functions, the algorithm may only reach a locally optimal region by the end of runtime. This is shown in Fig. 5, with designs the CIAAs produced that have especially high FOS values, over twice that required for the original design and higher than some of the representative CIAA-generated designs in Fig. 3. These designs still failed most of the adversary attacks, indicating that a similarly equipped adversary would likely be able to destroy them despite their high FOS. Such designs may appear beneficial due to high FOS, indicating the need to observe and test each objective component of completed designs and to track algorithm breadth of search. In seeking the application of this work to other problems, conditions must be directly modeled to design and test against via new sets of adversary grammars and design goals that capture different adversarial challenges to overcome.

Sensitivity analysis of adversary parameters indicates diminishing benefits of large adversarial team sizes and moves per designer move, though the design process becomes unstable and fails to converge at very small team sizes as the designer simply optimizes the truss against the small number of attacks it faces at its current step, blind to other vulnerabilities it may introduce by doing so. Computational cost is found to grow quickly with increased adversarial settings, however, resulting in a desired range of settings that balance their benefits and computational costs. A full sensitivity analysis with the exact settings and expanded results can be found in Rismiller, et al. [23].

4 Discussion

The primary design difference introduced when considering adversaries is the addition of structural redundancy. Because the adversaries are limited in how many members they can destroy, this redundancy ensures that the truss remains well-connected enough to sustain any loads the adversary then applies without collapsing, whereas designs created without considering these attacks can be destroyed with a single key member destruction. This design difference is driven by the interplay between designer and adversaries throughout the design process.

The agents as-presented focus on resistance to worst-case scenarios by taking the worst value over all adversary attacks. Future work could explore the consideration of probabilistic distributions of attacks as well. While the CIAAs create designs that resist the attacks that are modeled during their design, the designs are less efficient when applied to the normal load case, similar to findings in other literature, where robustness was achieved at the cost of optimality for the original purpose [24–27]. Such cases occur due to the additional objectives the designer must fulfill which prevents them from optimizing only for their original case. The agents' computational costs may also restrict them to problems that are impractical to analyze by other means.

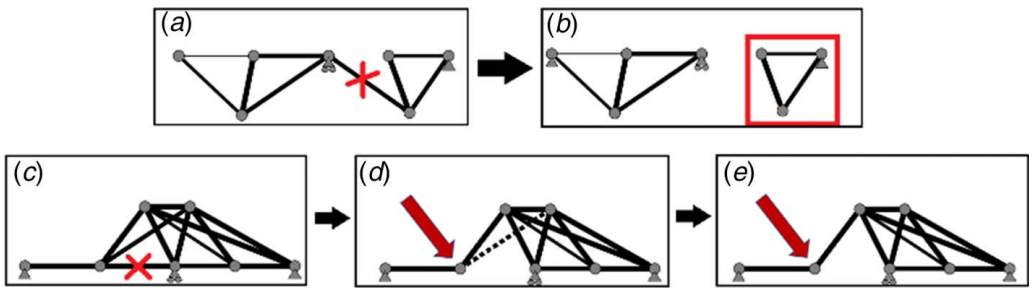

Fig. 4 (*a*) Example CISAT-generated truss attacked with member destruction, indicated by crossed member, (*b*) resulting unstable structure, (c) example CIAA-generated truss with a similar attack, (*d*) resulting structure with adversary load and induced member failure, and (e) final stabilized structure

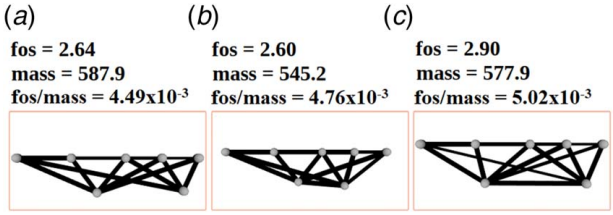

Fig. 5 Trusses produced with high factors of safety that did not survive all adversary attacks. These trusses survived (*a*) 25%, (*b*) 15.1%, and (c) 50% of attacks.

When run for a finite amount of time, simulated annealing often only approaches the exact optimal solution, rather than reaching it [28]. Because CIAAs use SA, and the limited number of iterations to design in, the design process will likely not reach an exact Stackelberg equilibrium. Instead, it will reach an approximate, or ϵ equilibrium, in which players deviating from their current strategies can make improvements by no more than a small value ϵ [29]. Because of this, resulting designs are not guaranteed to resist conditions not considered in this paper, and may require more extensive testing. Exact equilibria theoretically exist within this method, however, and more design iterations allow the agents to more effectively approach them at an additional computational cost. As detailed in the system description, at equilibrium the adversaries will have converged to the best attack plan for the given design and have no reason to deviate, while the designer cannot improve on their design without introducing a vulnerability that the adversaries can exploit. Therefore, no player can benefit from deviating in any sequence, fulfilling the conditions of Stackelberg equilibria.

We anticipate the work presented can apply to any configuration design problem. Previous work has shown similarities in the behavior and characteristics of problem-solving spaces for a variety of configuration problems including wave energy converters, HVAC, and fluid networks [30,31]. To extend this method to these problems, defining adversary actions against the solution that the designer is configuring, which may include component failures or abnormal conditions, will be necessary.

5 Conclusion

This paper introduces Cognitively Inspired Adversarial Agents capable of considering and accounting for reactions when making design changes. The resulting agents retain team-friendly cognitive problem-solving properties and design search pursued with simulated annealing, with consideration of problem scenarios added through adversary agents that follow Stackelberg Game conditions to attack the evolving design. CIAAs were applied to truss structures in this paper, but this method should apply to any configuration design problem that can be described by designer and adversary grammars with appropriate agent objective functions and problem structures. Analysis of unique features of the adversarially-generated designs created in these problems can then provide insight into designing against modelled conditions.

Acknowledgment

This work was supported by the Defense Advanced Research Projects Agency under Cooperative Agreement N66001-17-2-4064. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the view of the sponsor.

References

[1] McComb, C., Cagan, J., and Kotovsky, K., 2015, "Lifting the Veil: Drawing Insights About Design Teams From a Cognitively-Inspired Computational Model," Des. Stud., **40**, pp. 119–142.
[2] Bonabeau, E., 2002, "Agent-Based Modeling: Methods and Techniques for Simulating Human Systems," Proc. Natl. Acad. Sci. USA, **99**(Suppl. 3), pp. 7280–7287.
[3] Bertsimas, D., and Tsitsiklis, J., 1992, "Simulated Annealing," Stat. Sci., **8**(1), pp. 10–15.
[4] Gabrel, V., Murat, C., and Thiele, A., 2014, "Recent Advances in Robust Optimization: An Overview," Eur. J. Oper. Res., **235**(3), pp. 471–483.
[5] Park, G.-J., Lee, T.-H., Lee, K. H., and Hwang, K.-H., 2006, "Robust Design: An Overview," AIAA J., **44**(1), pp. 181–191.
[6] Cagan, J., and Williams, B. C., 1993, "First-Order Necessary Conditions for Robust Optimality," ASME Advances in Design Automation, Albuquerque, NM, Sept, 19-22, 1993, Vol. 1, pp. 539–549.
[7] Lee, K. H., and Park, G. J., 2001, "Robust Optimization Considering Tolerances of Design Variables," Comput. Struct., **79**(1), pp. 77–86.
[8] Changizi, N., and Jalalpour, M., 2017, "Robust Topology Optimization of Frame Structures Under Geometric or Material Properties Uncertainties," Struct. Multidiscip. Optim., **56**(4), pp. 791–807.
[9] Parkinson, A., Sorensen, C., and Pourhassan, N., 1993, "A General Approach for Robust Optimal Design," ASME J. Mech. Des., **115**(1), pp. 74–80.
[10] Kang, Z., and Bai, S., 2013, "On Robust Design Optimization of Truss Structures With Bounded Uncertainties," Struct. Multidiscip. Optim., **47**(5), pp. 699–714.
[11] Koch, P. N., Yang, R. J., and Gu, L., 2004, "Design for Six Sigma Through Robust Optimization," Struct. Multidiscip. Optim., **26**(3–4), pp. 235–248.
[12] Li, M., Azarm, S., and Aute, V., 2005, "A Multi-objective Genetic Algorithm for Robust Design Optimization," GECCO 2005—Genetic and Evolutionary Computation Conference, Washington, DC, June 25–29, pp. 771–778.
[13] Kolev, S., 2016, "Market Structure and Equilibrium, J. Hist. Econ. Thought, **38**(4), pp. 557–560.
[14] Chen, W., and Lewis, K., 1999, "Robust Design Approach for Achieving Flexibility in Multidisciplinary Design," AIAA J., **37**(8), pp. 982–989.
[15] Zhu, K., Hossain, E., and Anpalagan, A., 2015, "Downlink Power Control in Two-Tier Cellular OFDMA Networks Under Uncertainties: A Robust Stackelberg Game," IEEE Trans. Commun., **63**(2), pp. 520–535.
[16] Shiau, C. S. N., and Michalek, J. J., 2009, "Optimal Product Design Under Price Competition," ASME J. Mech. Des., **131**(7), p. 0710031.
[17] Liu, X., Du, G., Jiao, R. J., and Xia, Y., 2018, "Co-Evolution of Product Family Configuration and Supplier Selection: A Game-Theoretic Bilevel Optimisation Approach," J. Eng. Des., **29**(4–5), pp. 201–234.
[18] Hernandez, G., Seepersad, C. C., and Mistree, F., 2002, "Designing for Maintenance: A Game Theoretic Approach," Eng. Optim., **34**(6), pp. 561–577.
[19] Lewis, K., and Mistree, F., 1998, "Collaborative, Sequential, and Isolated Decisions in Design," ASME J. Mech. Des., **120**(4), pp. 643–652.
[20] Nash, J., 1950, "Non-cooperative Games," Ann. Math., **54**(2), pp. 286–295.
[21] Simaan, M., 1977, "Equilibrium Properties of the Nash and Stackelberg Strategies," Automatica, **13**(6), pp. 635–636.
[22] Triki, E., Collette, Y., and Siarry, P., 2005, "A Theoretical Study on the Behavior of Simulated Annealing Leading to a New Cooling Schedule," Eur. J. Oper. Res., **166**(1 Spec. Iss.), pp. 77–92.
[23] Rismiller, S. C., Cagan, J., and McComb, C., 2020, "Stochastic Stackelberg Games for Agent-Driven Robust Deisgn," ASME IDETC—Design Automation Conference, DETC2020-19197, St. Louis, MO, Aug. 16-19, 2020.
[24] Sandgren, E., and Cameron, T. M., 2002, "Robust Design Optimization of Structures Through Consideration of Variation," Comput. Struct., **80**(20–21), pp. 1605–1613.
[25] Sun, G., Zhang, H., Fang, J., Li, G., and Li, Q., 2018, "A New Multi-objective Discrete Robust Optimization Algorithm for Engineering Design," Appl. Math. Model., **53**, pp. 602–621.
[26] Holmberg, E., Thore, C. J., and Klarbring, A., 2017, "Game Theory Approach to Robust Topology Optimization With Uncertain Loading," Struct. Multidiscip. Optim., **55**(4), pp. 1383–1397.
[27] Laporte, G., Mesa, J. A., and Perea, F., 2009, "A Game Theoretic Framework for the Robust Railway Transit Network Design Problem," Transp. Res. Part B: Methodol., **44**(4), pp. 447–459.
[28] Mitra, D., Romeo, F., and Sangiovanni-Vincentelli, A., 1985, "Convergence and Finite-Time Behavior of Simulated Annealing," Proceedings of 24th IEEE Conference on Decision and Control, Ft. Lauderdale, FL, Dec. 11–13, pp. 761–767.
[29] Vazirani, V. V., Nisan, N., Roughgarden, T., and Tardos, É, 2007, *Algorithmic Game Theory*, Cambridge University Press, Cambridge, UK.
[30] Puentes, L., Mccomb, C., and Cagan, J., 2018, "A Two-Tiered Grammatical Approach for Agent-Based Computational Design," Proceedings of the ASME 2018 International Design Engineering Technical Conferences & Computers in Engineering Conferences, Quebec City, Canada, Aug. 26–29.
[31] McComb, C., Cagan, J., and Kotovsky, K., 2017, "Optimizing Design Teams Based on Problem Properties: Computational Team Simulations and an Applied Empirical Test," ASME J. Mech. Des., **139**(4), p. 041101.