A Morphogenetically Assisted Design Variation Tool*

Aaron Adler[1], Fusun Yaman[1], Jacob Beal[1], Jeffrey Cleveland[1], Hala Mostafa[1], Annan Mozeika[2]
[1]Raytheon BBN Technologies, Cambridge, MA 02138, [2]iRobot Corporation, Bedford, MA 01730
{aadler, fusun, jakebeal, jcleveland, hmostafa}@bbn.com, amozeika@irobot.com

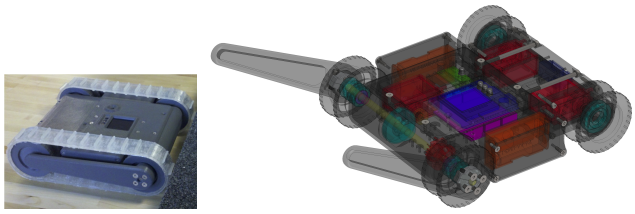
Abstract

The complexity and tight integration of electromechanical systems often makes them "brittle" and hard to modify in response to changing requirements. We aim to remedy this by capturing expert knowledge as functional blueprints, an idea inspired by regulatory processes that occur in natural morphogenesis. We then apply this knowledge in an intelligent design variation tool. When a user modifies a design, our tool uses functional blueprints to modify other components in response, thereby maintaining integration and reducing the need for costly search or constraint solving. In this paper, we refine the functional blueprint concept and discuss practical issues in applying it to electromechanical systems. We then validate our approach with a case study applying our prototype tool to create variants of a miniDroid robot and by empirical evaluation of convergence dynamics of networks of functional blueprints.

Introduction

Electromechanical systems, such as robots, vehicles, or consumer electronics, tend to be brittle in their design, particularly as the complexity of a system's design increases. Once a system has been constructed, it is difficult to modify the design without a vast number of consequences that can be unpredictably difficult and costly to address. Animals, on the other hand, are extremely complex, yet adapt gracefully as they grow and develop, with many feedback loops acting together to make changes that maintain the integration of the organism as a whole. Indeed, the flexibility and dynamicism of integration appear to be key enablers of the evolution of biological life (Carroll 2005; Kirschner and Norton 2005).

We aim to enable such adaptability in the design of engineered systems, so that when a designer modifies one element of a design, the rest of the design automatically adjusts to compensate. *Functional blueprints (FBs)* (Beal 2011) are a proposed approach to adaptability inspired by biological development, which capture expert knowledge by specifying a design through behavioral goals and a method for adjusting the structure when those goals are not met. In this paper, we refine the FB framework and apply it to the problem of electromechanical design, constructing an interactive design tool that uses functional blueprints to maintain integration of a design as it is being modified by a user. We validate this approach with a case study applying our tool to create a miniDroid variant that is capable of climbing a step over five times higher than the original specification. Finally, we test the generalizability of FBs by comparing against genetic algorithms on random networks of design constraints.



(a) miniDroid (b) CAD model

Figure 1: Even relatively simple electromechanical designs like this miniDroid robot frequently contain a large number of tightly integrated components.

Motivating Example: miniDroid

Consider an electromechanical design, such as the *miniDroid* robot shown in Figure 1: a small tracked robot capable of climbing over obstacles up to a certain height using its flippers. Even in this relatively simple design, tightly integrated components mean small changes frequently cause a cascade of other changes to preserve design functionality. For example, consider this cascade in a miniDroid:

1. The user increases the expected height of an obstacle.
2. Climbing higher requires a longer flipper and body.
3. Mobility of a larger robot requires stronger motors.
4. Endurance with stronger motors requires larger batteries.
5. Larger batteries add more mass, which may require different slightly larger motors, etc.

Note that some functions, such as the ability to climb a step, can be evaluated in simulation, but have no closed-form

*Work sponsored by DARPA DSO under contract W91CRB-11-C-0052; the views and conclusions contained in this document are those of the authors and not DARPA or the U.S. Government.

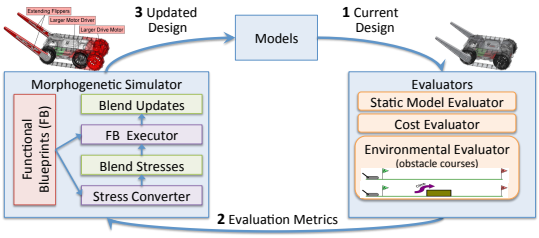


Figure 2: The MADV architecture iteratively adapts a design against a collection of functional blueprints (FBs) using a three stage loop: (1) evaluating system behavior, (2) blending the action of all of the FBs in a morphogenetic simulator, and (3) producing an incrementally modified design.

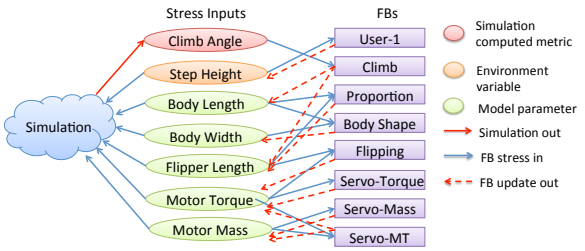


Figure 3: Parameter/FB relations in miniDroid case study.

analytical evaluation. The simulation requirement precludes parametric design, and makes it prohibitively costly to use most search or optimization techniques. Our aim is to radically reduce the number of evaluations needed for redesign by capturing relationships between behavior and design as *functional blueprints*. These relationships and user-initiated specification changes guide the design modifications.

Framework for Design Variation

This section briefly reviews the functional blueprint concept, then describes the Morphogenetically Assisted Design Variation (MADV) architecture we have developed for applying it to problems in electromechanical design. The MADV architecture and the miniDroid case study were developed during the first year of a DARPA-funded effort.

Functional Blueprints (FBs)

FBs (Beal 2011) are inspired by regulatory processes in natural morphogenesis, where the physical form of a developing organism is incrementally created with influence from the dynamically changing requirements of other systems. FBs consist of four elements: 1) a *system behavior* that degrades gracefully across some range of viability, 2) a *stress metric* quantifying the degree and direction of stress on the system, 3) an *incremental program* that relieves stress through growth, shrinkage, or other structural change, and 4) a program to construct an initial viable *minimal system*.

The basic idea is to keep a system always viable, like a growing animal or an evolving species. The minimal system gives a viable starting point. Since behavior degrades gracefully, it is possible to detect when design elements start to become problematic. The *stress* metric, a domain-specific measure of system adequacy, is fed as a diagnostic signal to the incremental program, which repairs it, e.g., when the amount of power drawn exceeds a threshold, the system becomes stressed and increases battery capacity. A collection of FBs thus uses stress as a coordinating signal to determine which subsystems can safely be modified at any given time. When nothing is stressed, any element can be incrementally modified; when functionality begins to degrade, stress rises and is propagated through the network of FBs as each FB acts locally to reduce stress.

The correctness and completeness of the approach depends on the set of FBs. In general, however, this approach is expected to be applicable if the viable space is continuous, well-connected, and all FBs have graceful degradation. Assuring graceful degradation can be difficult however, if a stress function is non-monotonic or has a non-deterministic evaluation (e.g., from an unreliable simulation).

Even when all FBs are well-behaved, reaching a no-stress solution is not guaranteed, because it may not exist or may be blocked by intervening non-viable space. In this case, however, FBs will still find the closest accessible solution and indicate with stress which design aspects are limiting.

MADV Architecture

One of our goals is to enable users, who may be knowledgable about a desired functionality but novices at electromechanical design, to create a functional design variant. To enable this, we allow users to indirectly control design variation through modifications of the simulated test environment, e.g., changing the height of an obstacle to climb over in Figure 4. Unlike the purely structural "tissue" models studied in (Beal 2011), electromechanical designs comprise many discrete components and parameters. Additionally stress in electromechanical devices may not be computable from design parameters, and instead is measured indirectly through simulations. Our MADV architecture (Figure 2) coordinates a set of design FBs that operate on real-valued parameters.

MADV is based on a three-phase design modification loop, which runs until the design converges to a stress minimum. First, the functionality of the current design is analyzed by a set of *evaluators*, which measure how well a design accomplishes its goals. The input to each evaluator is a subset of the current design parameters (e.g., robot model, testing environment) and the outputs are a set of metrics (e.g., time to do a required task, body angle). We have identified four classes of FBs (and associated evaluators) for electromechanical design: closed form, quantized component, simulation-driven, and user-command. Examples of these FBs for the miniDroid are shown in Figure 3 and are described in detail in the next section. Note that these evaluators can potentially be very complex and costly to compute, particularly the simulations.

The second phase is the *morphogenetic simulator*. Each FB's stress function converts evaluation metrics to stresses. An FB may receive metrics (and thus stresses) from multiple evaluators: these are blended together into a single stress

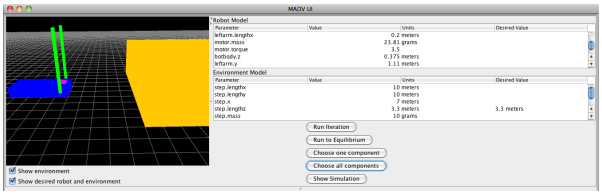


Figure 4: MADV UI allows design or environment changes.

measure, which is input to the FB's incremental program to generate update recommendations for parameters. Each parameter, in turn, may receive update recommendations from multiple functional blueprints, which must be blended together to determine how the parameter is updated.

These blends of stress and updates are thus key to design adaptation. At present, MADV uses the same heuristic function for both stress and parameter updates:

$$V = \text{sign}(\sum_i V_i) \cdot \max_i (V_i \frac{\sum_i V_i}{\sum_i \mid V_i \mid})$$

where the V_is are the inputs and V is the blended value. This blending function preserves the direction of the sum of the values, resulting in a value that is a fraction of the largest value in that direction. It is designed to result in smaller values when blending values with opposite signs, but preserves the maximum value when all the values have the same sign. For example, if $V_i = \{0, 1, 3, 6, 10\}$ then $V = 10$; if $V_i = \{-6, -3, -1, 5\}$ then $V = -2$

Finally, a new, incrementally modified design is produced from the updated parameters. For the work in this paper, we use a simple parametric adjustment; the same framework, however, can support more complex layout changes like those in (Beal et al. 2012). With an appropriate choice of stress metrics, incremental updates, and blending function, a system of FBs can be applied to make incremental adjustments to the design of an electromechanical system.

Our implementation of the MADV architecture is not tied to the miniDroid case study: the software is highly modular, and intended to simplify the reuse of FBs: FBs, parameters, and evaluator settings are all specified in XML files, and evaluators are implemented as modular Java objects that can enable arbitrarily complex problem-specific evaluation (e.g., miniDroid simulations were invoked via remote procedure calls). This supports a user interface (Figure 4) that allows the user to adjust both design and environment attributes (e.g., step height, flipper length), to observe stress levels (both per FB and total), and to visualize the design as it is modified by successive iterations.

Case Study: miniDroid

We applied FBs and the MADV architecture to the problem of constructing a miniDroid variant that can climb over much taller obstacles. For our case study, we selected a subset of critical miniDroid functions, including climbing over obstacles, flipping itself over, and agile maneuvering. We then consulted with a robotics design expert to identify seven key parameters affecting these functions and eight FBs for evaluating them. The relations between FBs and parameters are shown in Figure 3 (including the user's desire to climb a higher obstacle). Note that the simulation for step climbing is affected by all of the design parameters.

To evaluate this approach, we: 1) created FBs controlling this subset in four categories as described below, 2) applied MADV to generate a large scale variation, and 3) evaluated scalability by considering the complexity of the complete miniDroid design. In the following subsections, we discuss each phase in turn, as well as explaining in detail the four different classes of functional blueprints and the relative effort required to create them.

Creation of Functional Blueprints

Closed-Form FBs Closed-form FBs are those whose evaluation can be done analytically on the basis of some set of equations. These FBs are generally simple to create with the assistance of a domain expert and very fast to evaluate. Three FBs in our miniDroid case study are closed-form. The *Body Shape* FB constrains the miniDroid's length-to-width ratio (reflecting a heuristic for assessing the agility of a tracked vehicle) and is evaluated by calculating this ratio. The *Proportion* FB maintains a ratio between flipper length and body length to avoid problems with sensing and step-climbing. The *Flipping* FB ensures that the flipper motor has enough torque to lift the robot body. Other examples of closed-form FBs might include regulation of robot cost, battery endurance, and the strength of the flipper drive shaft.

Quantized-Component FBs Quantized-component FBs handle design parameters that are chosen as a set from a restricted library of available component variants. This is important for electromechanical design because most designs rely on some Commercial Off The Shelf (COTS) components like servos, microcontroller boards, and batteries. An FB of this type is constructed by collecting a large set of available instances of the component class and their parameters (e.g., from vendor data sheets), then approximating an envelope around this class of components in parameter space. The envelope edges act as a closed-form FB to keep parameter values near available instances. While design stress is high, parameters can be anywhere within the continuous envelope; when the design converges to a low-stress state, parameters are set to the nearest available instance. The graceful degradation condition ensures that quantizing a component is unlikely to significantly impact design viability. These are harder to construct than a closed-form FB, due to the need to gather parameter data, but are similarly fast to evaluate.

Our miniDroid case study uses one quantized-component FB: the *Servo* FB relates the torque and mass of servo motors available to drive miniDroid flippers and wheels. Figure 5 shows the mass and torque dimensions and approximating envelope (implemented as three closed-form FBs: Servo-Mass, Servo-Torque, and Servo-MT (Mass Torque)).

Simulation-Driven FBs Simulation-driven FBs are generally used to evaluate complex functions that interact with many different aspects of the design and which have no closed-form evaluation. Use of simulation makes these FBs

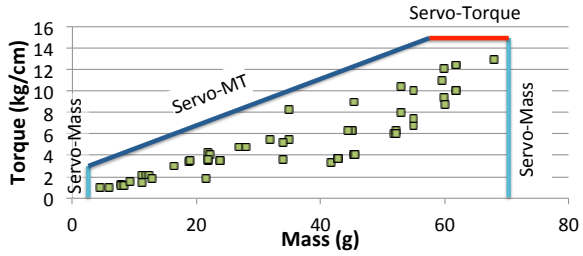


Figure 5: Key properties of a servo library are captured using functional blueprints that become stressed if the design ever needs a servo outside the envelope of available servos.

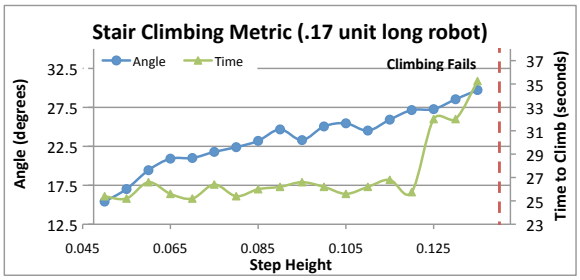
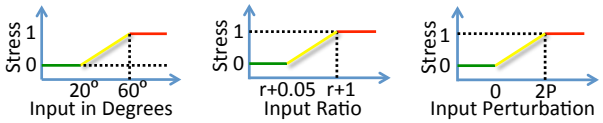


Figure 6: Time and angle as possible stress metrics for climbing function. Only angle degrades gracefully.

the most challenging to define and most computationally expensive to evaluate. We use one simulation-driven FB for the miniDroid: the *Climb* FB maintains the miniDroid's ability to climb over obstacles. This is evaluated by a simulation in which the robot climbs over a box-shaped step, implemented with ROS (Robot Operating System) (Quigley et al. 2009) using the Gazebo simulator.

When considering the task of a robot climbing over an obstacle, there are multiple reasons climbing may fail, e.g., insufficient friction between the robot's tracks and the obstacle's surface, or the robot may be too short. There is no direct measurement to assess how stressed the robot is while climbing. Our robotics expert indicated that climbing time and body angle at the "critical point" might be good metrics for the stress computation (our climbing algorithm defines the critical point to be when the flippers are parallel to the body). To know when the design is stressed, we need a metric that gracefully degrades as the robot begins to have trouble climbing the step. We ran simulations with fixed robot size and increasing step height. Figure 6 shows the behavior of the two possible stress metrics. From this data, angle appears to be a better metric than time because it progresses more gradually toward failure. Additionally, the relationship between step height and climb time is less direct: requiring the robot to complete the climb in the same amount of time (thus move faster), would introduce stress on components not directly related to climbing and create the need for extremely rapid movements at larger scales. We therefore defined the stress function of the climb FB based on the robot angle measurement, as shown in Figure 7(a).



(a) Step Climb (b) Ratio Limit (c) Test Driver

Figure 7: Example FB stress functions: green is zero stress, yellow is stressed, and red is non-viable.

User-Command FBs User-command FBs are used to prevent parameter changes initiated by the user from making a design non-viable. Instead of changing the parameter as specified by the user, MADV instead creates a temporary FB that incrementally shifts a design or evaluator parameter towards the specified value. Parametric representation of the design and the simulated environments thus enable a novel user interaction, in which the user can observe and guide a design as it shifts toward a new specification.

In this case study, the user requests one design change, however there is no limit to the number of changes the user can request. In this case, the user increases the initial 10cm high step to 55cm, resulting in one user-command FB. If this were implemented at once, the robot could not climb at all; incremental change, however, facilitates the adaptation.

Generating a Large-Scale miniDroid Variation

We tested the ability of our FBs to generate viable variations with a user-increase of step height from 10cm to 55cm. Faced with this challenge, MADVs successfully adapts the miniDroid design over the course of 75 iterations (all of which are functional intermediate designs), until the variant can climb the larger step and no design function is stressed. Figure 8 shows the changes in the stress levels and parameters as the system converges to a new equilibrium, with some parameters changing more than 5x with respect to their initial values. Note that the stress on various FBs rises and falls over time as the design encounters different limiting constraints, and that some FBs never become stressed at all.

Running the system to generate this new variant took about an hour. The bottleneck is the step-climbing simulation (everything else is executed in negligible time), due to the approximation used for simulating tread locomotion. This highlights the importance of using an approach like MADV to guide the incremental process of design variation.

Scalability and Discussion

For an initial evaluation of scalability, we consulted with our robotics expert to create a model of the full miniDroid containing all design features of components larger than 1 cm^3 in volume. Identifying all parameters and FBs took 3 hours and yielded 66 parameters linked by 29 FBs (21 closed-form, 3 quantized-component and 5 simulation-driven).

We thus see that, for the case of the miniDroid, the FB approach can be used to assist in generating a design variant, and is scalable in terms of the effort required to create a complete set of FBs. An important limitation in the current implementation, however, is that user-command FBs targeting

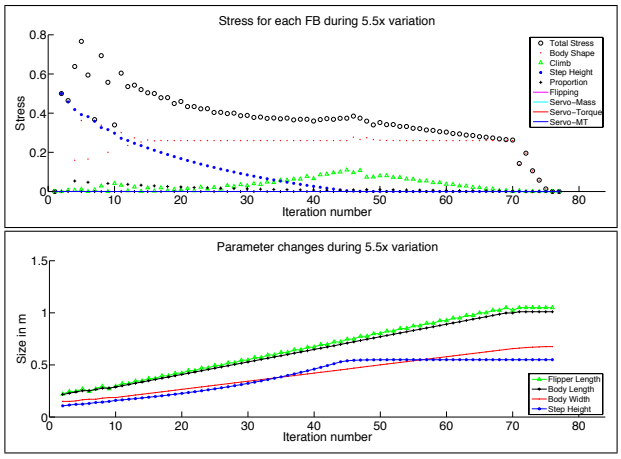


Figure 8: Stress (top) and parameter size (bottom) over time as the miniDroid design returns to a zero stress state after increasing the step height from 10cm to 55cm.

environment parameters (e.g., step height) are operating in an open-loop fashion. The reason for this is that we want the environment to be independent of the robot's capabilities. Additionally, there may be many simulations-based FBs, for example a rubble-field simulation, which means that the parameter set for the environment is open. The user-command FBs must therefore be very conservative, or else risk driving the design into non-viability (e.g., because the step is growing faster than the robot can adapt). This could be remedied by limiting the updates of user-command FBs based on the maximum stress experienced by any other FB.

Generalization and Comparison of FBs

Having successfully used our MADV architecture to adapt the miniDroid design to varying conditions, we posed the questions of 1) how our approach generalizes and scales, and 2) how does it compare to standard approaches like Genetic Algorithms (GAs). Since GAs evaluate the fitness of large numbers of individuals per generation, which in the miniDroid case involves invoking the (computationally expensive) simulator, it would be infeasible to use GAs in the context of our case study. Instead, we randomly generated abstract networks of design variables and FBs.

Experimental Setup

We consider a set of n design attributes and k FBs. Each FB operates over a random pair of attributes with a desired ratio set to a random value in [0.5, 2]. The stress function in Figure 7(b) is used with the ratio between the two attributes as its input. When stress is non-zero (in $(0, 1]$), the FB incrementally adjusts both attribute values by $stress * 2$ in the direction that will reduce stress, e.g., if the ratio is high, the numerator attribute will be decreased and denominator attribute increased. An example network is shown in Figure 9.

One attribute's value is perturbed with a user-command FB that prescribes an increase at rate d towards a desired value of $1 + P$ times the original value. The stress function (Figure 7(c)) takes the normalized difference, $\frac{desired-current}{current}$, where $current$ is the current attribute value and $desired$ is the goal. The incremental update when stress is non-zero is $min(|desired - current|, current * d)$. We call d the *drive rate*. Unless otherwise mentioned, we use a drive rate of $d = 0.005$, a goal of $P = 1$, and run each parameter configuration 20 times, each time running at most 1000 iterations of the design modification loop.

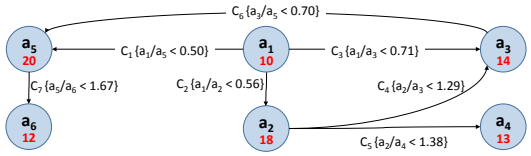
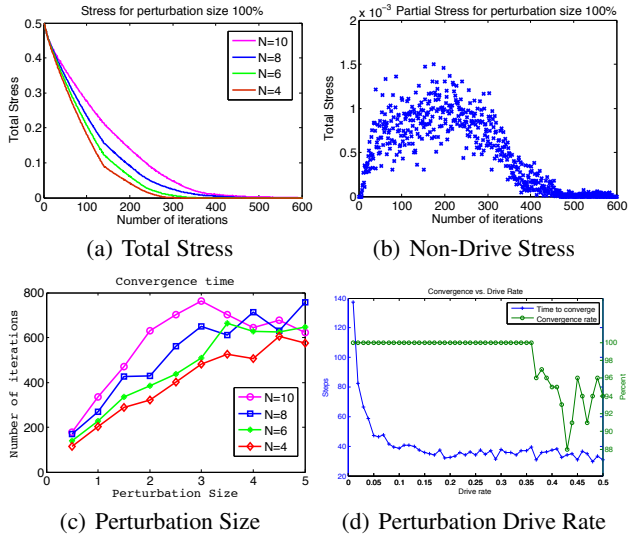


Figure 9: Example of a randomly generated functional blueprint network with six attributes and seven blueprints.



(a) Total Stress (b) Non-Drive Stress

(c) Perturbation Size (d) Perturbation Drive Rate

Figure 10: The main component of the mean total stress (a) is the driver used to inject the perturbation: the network of ratio constraints rapidly adapts, keeping the mean stress over all experiments very low (b). Convergence is robust against a wide range of perturbation sizes (c) and drive rates (d).

Results and Discussion

We begin by examining the behavior of stress over time, considering four different random network sizes up to 20 FBs: (n, k) equal to $(4, 4)$, $(6, 7)$, $(8, 12)$, and $(10, 20)$. In these experiments, the FBs always converge, leaving the perturbed attribute at its new value and all other attributes at adjusted zero-stress values. Figures 10(a) and 10(b) show the total stress over time and stress only in the ratio constraint network, respectively. For the latter, we see a "spiky" pattern of small amounts of stress appearing and rapidly vanishing, showing that these FBs are adapting to the driving perturbation across a wide range of sizes. These figures show highly effective adaptation, where stress from the perturbation is dispersed as quickly as it is injected into the network.

This rapid and robust adaptation holds across a wide range of perturbation sizes, drive rates, and number of FBs. Fig-

Perturbation size 1	| Initial - Final |	# Iterations	Time (s)	Stress	Convergence
FB	**22**	**310**	**0.01**	**0**	**100%**
GA w/o stress (FF1), pop 50	48	756	0.12	0.65	35%
GA w/ stress (FF2), pop 50	80	714	0.52	0.2	40%
GA w/o stress (FF1), pop 500	55	714	2.65	0.14	40%
GA w/ stress (FF2), pop 500	88	323	2.8	0.05	85%

Table 1: FBs converge faster and more reliably than GAs.

ures 10(c) and 10(d) show variation in convergence time with perturbation size ranging from $P = 0.5$ to 5 and for perturbation drive rate ranging from $d = 0.01$ to $d = 0.5$ (100 iterations per d value), respectively. Only when d is extremely high, changing attribute value by more than 35% in a single iteration, does the system fail to converge.

GA Comparison

We compared FBs to GAs on the same set of random design constraint networks on a machine with a 2.2GHz Intel Core i7 processor and 8GB of memory. While FBs use stress signals received from each constraint to produce a new potential solution from an old one, GAs use a fitness function and natural selection to produce new generations of potential solutions. We used two fitness functions: FF1 is a function of distances from the desired and initial solutions, and the sum of constraint violations; the more informed FF2 is a function of the stress inputs used by the FBs. We implemented our GA on top of JGAP (JGAP 2006) and experimented with different settings of GA parameters like population size, mutation rate, and magnitude, with the best results summarized here. The number of iterations for finding a solution is bounded by 1000 for both algorithms.

Table 1 shows that the FBs search is much more focused, and thus more efficient, than the search done by the GA. Even with the better informed fitness function (FF2), FBs are still superior to GAs. FBs achieved a convergence rate of 100% as opposed to the best GA result of 85%, even though each FB iteration involves less work than a GA iteration (a generation). The FBs also converge to a solution that is stress-free, which is not the case for the GA. FBs solved the networks 280 times faster than the GA with best convergence rate. Additionally, the solutions reached by the FBs were much closer to the initial solution. This increased efficiency comes at the price of being limited to the viable design spaces and prevents radically different (and potentially superior) designs from being discovered.

Related Work

Constraint based local search (CBLS) engines such as Kangaroo (Newton et al. 2011) and Comet (Van Hentenryck and Michel 2005) are shown to be efficient in solving constraint satisfaction problems that are difficult for solvers based on constructive search. In CBLS the problem is formulated as constraint optimization where the objective is to minimize the constraint violation penalties, and this is accomplished through exploring a local neighborhood of parameter changes. MADV and CBLS shares two major properties: 1) stress in FBs are analogous to constraint violation penalty, and 2) the incremental update functions in FBs ensure the next parameter assignments will be in a neighborhood of the previous assignments. The major difference between the two approaches is that the update functions in FBs contain recipes on how to relieve stress by specifically identifying the parameters that need to be adjusted, as well as the direction of adjustment. FBs thus require more knowledge engineering, but require many less evaluations as they explore the parameter space, which is of critical importance when evaluation is costly, e.g., requiring simulation.

Systems engineering also considers the problem of guiding the development of systems with intricate and diverse relationships (Kossiakoff et al. 2011). Automated design exploration has been applied to the conceptual design stage where the "design" is largely schematic and not detailed. For detailed design exploration, many automated design exploration methods utilize evolutionary algorithms to generate novel designs. Fan, Wang, and Goodman describe a method of exploring a design space using evolutionary processes operating on a bond graph model. Koza et al. explore the design of analog circuits by means of genetic programming where the interfaces between interacting components is well defined and the components are limited to a single technical discipline.

Additionally, commercial modeling and simulation tools have been created to aid in design variation. For example the well-known simulation software developer, ANSYS, has parametric design exploration tools (ANSYS, Inc. 2013) to aid in design optimization. These tools are largely intended for parameterized structural exploration of single components such as optimizing the strength of a bracket, or fluid flow through complex geometry.

Other approaches to adaptive design of functional structures includes Werfel's work on distributed construction (Werfel and Nagpal 2007), and various projects in self-reconfigurable robotics e.g., (Stoy and Nagpal 2004; O'Grady, Christensen, and Dorigo 2009; O'Grady et al. 2010). These approaches, however, are generally intended for more homogeneous and loosely coupled systems and would be difficult to adapt to electromechanical design.

Contributions and Future Work

We have demonstrated that the functional blueprint concept can be applied to electromechanical design, providing assistive automation in the development of new design variants. Using our MADV architecture, we have presented a case study of the miniDroid robot in which seven key FBs are identified and used to produce a variant with parameters changing by more than 5x from the original. We have also demonstrated that under certain conditions FB networks converge and vastly outperform GAs.

These results represent a successful proof of concept for the MADV approach. Scaling up to practical application to complex designs will require improvements to both the software, including distributed parallel evaluator execution, support for more complex FB specifications, and automatic extraction of quantized-component envelopes; and to the theory of FB networks, including hierarchical components, dynamic adjustment of update size, adaptability of design "body plan," and improved convergence bounds.

References

ANSYS, Inc. 2013. ANSYS DesignXplorer. http://www.ansys.com/Products/Workflow+Technology/ANSYS+Workbench+Platform/ANSYS+DesignXplorer.

Beal, J.; Mostafa, H.; Mozeika, A.; Axelrod, B.; Adler, A.; Markiewicz, G.; and Usbeck, K. 2012. A manifold operator representation for adaptive design. In *GECCO 2012*.

Beal, J. 2011. Functional blueprints: An approach to modularity in grown systems. *Swarm Intelligence Journal* 5(3-4):257–281.

Carroll, S. B. 2005. *Endless Forms Most Beautiful*. W. W. Norton & Co.

Fan, Z.; Wang, J.; and Goodman, E. 2005. *Cutting Edge Robotics*. Germany: Pro Literatur Verlag. chapter Exploring Open-Ended Design Space of Mechatronic Systems, 707–726.

JGAP. 2006. JGAP - Java Genetic Algorithms Package: http://jgap.sf.net.

Kirschner, M. W., and Norton, J. C. 2005. *The Plausibility of Life: Resolving Darwin's Dilemma*. Yale University Press.

Kossiakoff, A.; Sweet, W. N.; Seymour, S.; and Biemer, S. M. 2011. *Systems Engineering Principles and Practice*. Wiley-Interscience, 2 edition.

Koza, J. R.; Bennett III, F. H.; Andre, D.; and Keane, M. A. 1998. Evolutionary design of analog electrical circuits using genetic programming. In *Adaptive Computing in Design and Manufacture*, 177–192. Springer.

Newton, M. A. H.; Pham, D. N.; Sattar, A.; and Maher, M. J. 2011. Kangaroo: An efficient constraint-based local search system using lazy propagation. In *Principles and Practice of Constraint Programming, 17th International Conference, CP*, 645–659.

O'Grady, R.; Christensen, A. L.; Pinciroli, C.; and Dorigo, M. 2010. Robots autonomously self-assemble into dedicated morphologies to solve different tasks. In *Autonomous Agents and Multiagent Systems*.

O'Grady, R.; Christensen, A.; and Dorigo, M. 2009. Swarmorph: Multi-robot morphogenesis using directional self-assembly. *IEEE Transactions on Robotics* 25(3):738–743.

Quigley, M.; Gerkey, B.; Conley, K.; Faust, J.; Foote, T.; Leibs, J.; Berger, E.; Wheeler, R.; and Ng, A. Y. 2009. ROS: an open-source robot operating system. In *Proceedings of the Open-Source Software workshop at the International Conference on Robotics and Automation (ICRA)*.

Stoy, K., and Nagpal, R. 2004. Self-reconfiguration using directed growth. In *Intl. Symposium on Distributed Autonomous Robotic Sys. (DARS)*.

Van Hentenryck, P., and Michel, L. 2005. *Constraint-Based Local Search*. MIT Press.

Werfel, J., and Nagpal, R. 2007. Collective construction of environmentally-adaptive structures. In *Int'l Conf. on Intelligent Robots and Sys.*