Distributed Adaptive Constrained Optimization for Smart Matter Systems

Markus P.J. Fromherz, Lara S. Crawford, Christophe Guettier, and Yi Shang
Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, CA 94304
{fromherz,lcrawford,cguettier,yshang}@parc.xerox.com

Abstract

The remarkable increase in computing power together with a similar increase in sensor and actuator capabilities now under way is enabling a significant change in how systems can sense and manipulate their environment. These changes require control algorithms capable of operating a multitude of interconnected components. In particular, novel "smart matter" systems will eventually use thousands of embedded, micro-size sensors, actuators and processors.

In this paper, we propose a new framework for a on-line, adaptive constrained optimization for distributed embedded applications. In this approach, on-line optimization problems are decomposed and distributed across the network, and solvers are controlled by an adaptive feedback mechanism that guarantees timely solutions. We also present examples from our experience in implementing smart matter systems to motivate our ideas.

Introduction

The remarkable increase in computing power together with a similar increase in sensor and actuator capabilities now under way is enabling a significant change in how systems can sense and manipulate their environment. These changes require control algorithms capable of operating a multitude of interconnected components. In particular, novel "smart matter" systems will eventually use thousands of embedded, micro-size sensors, actuators and processors. As an example, consider an air-jet paper mover, consisting of 576 actuators and 30K sensor pixels embedded into an active surface that moves a sheet of paper on an air bed by controlling each jet valve individually (Jackson *et al.* 2001). As another example, consider a hyper-redundant modular robot that consists of hundreds of interchangeable robotic modules connected into complex configurations, each module with its own actuated joint, sensors, processor, and communication (Yim 1994). As yet another example, consider materials with embedded, co-located force actuators and sensors for active vibration control (Chase, Yim, & Berlin 1999; Hogg & Huberman 1998) or damage identification (Wang & Chang 2000). Such systems will require control, sensing and diagnostic algorithms that are robust, scalable, and reconfigurable.

Constrained optimization is at the core of many planning, control, reconfiguration, and fault diagnosis applications (Bondarenko, Bortz, & Moré 1998). In fact, the development of optimal controllers and diagnostic routines for smart matter systems often starts with the formulation of a model of the system dynamics and an objective function to be optimized by the algorithm (e.g., the cost function of an LQR controller). For some systems, such as modular reconfigurable robots, only an on-line, model-based control approach is able to robustly handle the variety of non-standard objectives and constraints required (Fromherz, Hoeberechts, & Jackson 1999; Fromherz *et al.* 2001).

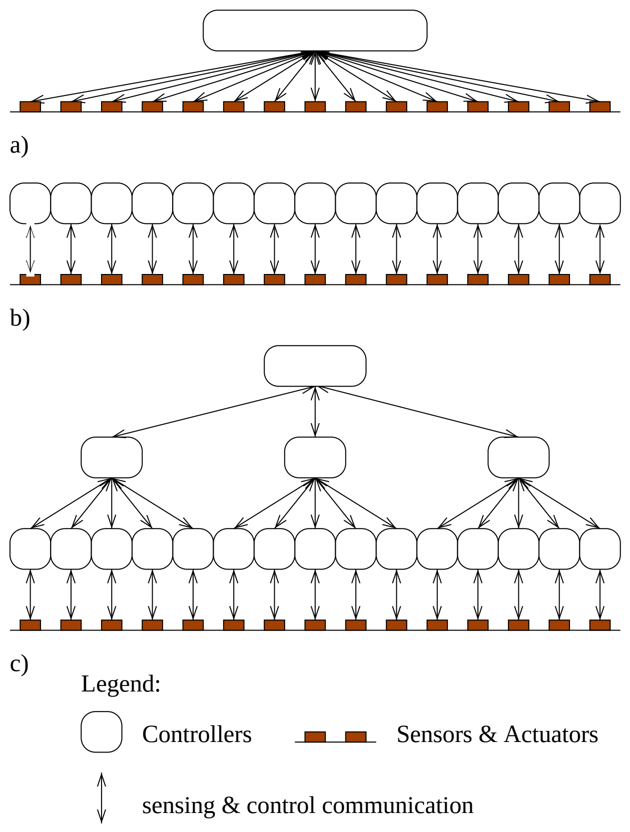


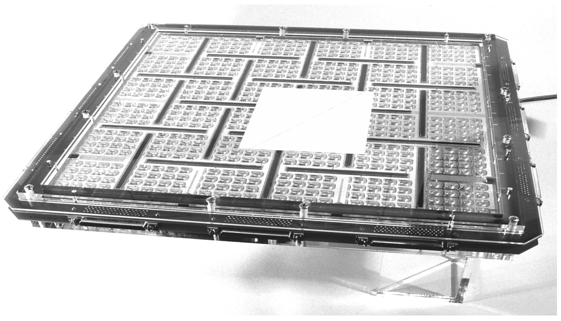
Figure 1: Sketches of a) centralized, b) decentralized, and c) hierarchical control organization.

Today, solutions to these control problems are usually at either one of two extremes: they are either completely centralized or completely decentralized. Centralized algorithms (Fig. 1a) provide optimal performance (taking into account the complete system behavior when controlling each actuator), but clearly do not scale to the large numbers of elements in smart matter systems. Completely decentralized (i.e., local) algorithms (Fig. 1b), on the other hand, scale well and make use of a system's distributed processing capabilities, but they typically result in poor performance (e.g., introducing large errors or resulting in inordinate actuation energies).

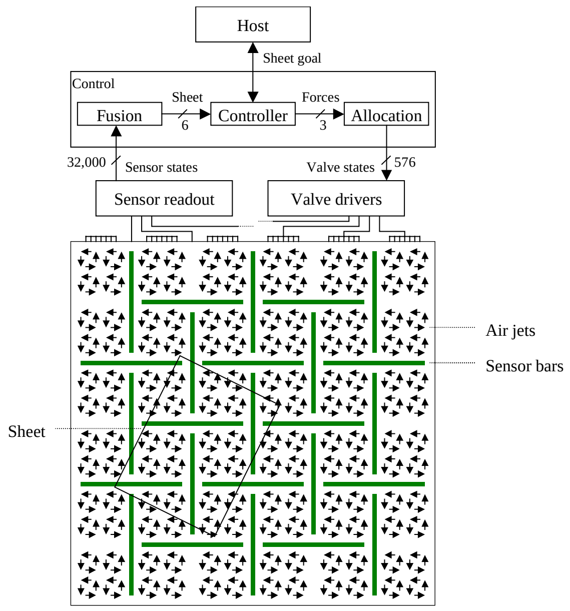
Clearly, *hierarchical approaches* (Fig. 1c) promise an intermediate solution: at various levels of the hierarchy, the algorithm takes into account groupings of system elements and their behaviors, while also being able to distribute the computation among a number of processing nodes. Consider for example the force allocation problem in the air-jet paper mover (Fig. 2) (Jackson *et al.* 2001; Fromherz & Jackson 2001), where the goal is to generate an optimal allocation for the air jets such that together they produce the desired forces and torques that will move the sheet of paper (see below for details). Our implemented approach is a self-similar, hierarchical decomposition of this task, where the actuators are partitioned into smaller and smaller groups (Fig. 3): at every level in the hierarchy, jets are combined into groups that act like "super jets" (each delivering forces that act on the paper), and no level has to know whether the next-lower level consists of individual jets or jet groups. Similar considerations have been made for other smart matter systems (Hogg & Huberman 1998).

From a model-based computing perspective, however, there are at least two major barriers to employing this hierarchical, distributed approach in real-world systems: (1) the decomposition of a centralized formulation into a hierarchical structure is non-trivial, as it often involves complex problem reformulations and the balancing of trade-offs between performance and resource uses (computing, communication) that have to be modeled explicitly; and (2) today's optimization / constraint solving algorithms are not ready for real-time, resource-constrained applications.

We have studied and developed smart matter applications for a number of years and found several suitable model-based control solutions (Fromherz, Hoeberechts, & Jackson 1999; Jackson *et al.* 2001). From this experience, we are proposing a new framework for an (on-line) adaptive constrained optimization service for distributed embedded applications. In this approach, model-based control and thus on-line optimization problems are decomposed and distributed across the network, and solvers are controlled by an adaptive feedback mechanism that guarantees timely solutions. The framework addresses the two barriers described above, i.e., problem scale and real-time constrained optimization. This paper discusses the two main components of our approach, decomposition and adaptive control of solving, in the following two sections. We also describe illustrative examples to motivate our ideas and end with conclusions and future work.



a)



b)

Figure 2: Air-jet system: a) Photograph of 35 cm × 35 cm air-jet paper mover module: 16-element arrays are flap valves and associated jets, black bars are sensor arrays. b) System architecture and layout of the board.

Problem Decomposition for Distributed Solving

Because of the large numbers of sensors, actuators, and computing elements in smart matter systems, we suggest that problem decomposition and distribution techniques are core to enabling and delivering a scalable and robust solver service for such systems. In particular, decomposition allows us to balance control requirements with communication and resource realities. For problems with thousands of variables and constraints, representing the large number of distributed but connected physical elements in large-scale smart matter systems, it is often both infeasible and unnecessary to solve the constrained optimization problem as a single problem on a single processor. Instead, our proposed approach is based on the idea that systems with tens or hundreds of thousands of sensors and actuators, whether discrete or continuous at the individual element level, approach a continuum

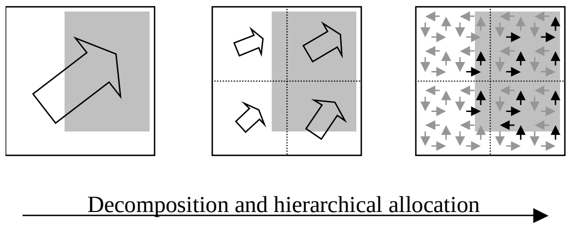


Figure 3: Decomposition of a force allocation problem into sub-problems on an active surface

in the limit and thus can be approximated by much lower-dimensional continuous models at higher levels. (This puts the conventional idea of hybrid systems on its head, where discrete models are usually at the top and continuous models at the bottom.) A second key insight is that such large-scale problems are often self-similar at multiple levels, as already illustrated above. Finally, we have observed that different groupings of elements can lead to widely different computational complexities at lower levels. Even if the hierarchical abstraction is fixed ahead of time, we can still decide dynamically which concrete actuators to group together, e.g., based on run-time information about available actuators.

Constraint-directed problem decomposition

We are investigating two schemes for automatically decomposing tasks on many-element systems. In a *constraint-directed approach*, we are developing structure analysis and abstraction techniques for decomposing and approximating large-scale constraint problems. The goal is to discover structure in constraint problems that can be exploited by solvers for the subproblems. For instance, we have found that maximizing symmetries in subproblems by grouping actuators that are interchangeable with respect to the constraints can make a dramatic difference. As shown in an example below, by using the problem's constraints to guide the grouping, solving subproblems at lower levels of the hierarchy can become trivial. In certain systems, it may even be possible to precompute the lower-level solving step and replace it by a constant-time table lookup. Problem characteristics such as symmetries seem to be common in systems where constraints arise out of (often uniform) physical characteristics. This proposed approach of structure analysis and abstraction can lead to considerable increases in efficiency with reasonable error bounds for problems in large-scale systems (Fromherz & Jackson 2001).

An example for problem decomposition

We illustrate the potential of constraint-directed decomposition on the air-jet paper mover introduced above. For example, the air-jet force allocation problem can be formulated as an optimization problem, attempting to achieve the desired total forces while minimizing actuation energy (Fromherz & Jackson 2001):

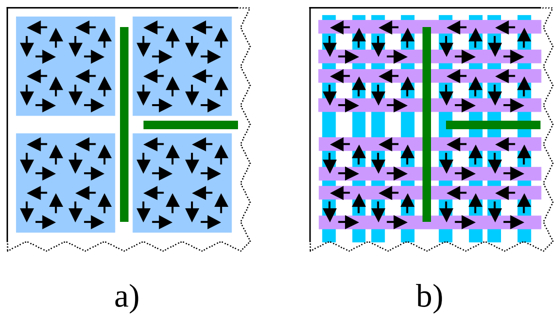


Figure 4: Aggregation heuristics applied to the air-jet table (excerpts): a) neighborhoods in geometric space (tiles); b) neighborhoods in geometric and force space (rows and columns)

$$\begin{array}{rl} \min_{f_{\mathrm{x}i},f_{\mathrm{y}i}} & \frac{1}{2}\sum_{i=1}^{N}\frac{f_{\mathrm{x}i}^2}{w_{\mathrm{x}i}^2}+\frac{1}{2}\sum_{i=1}^{N}\frac{f_{\mathrm{y}i}^2}{w_{\mathrm{y}i}^2} \\ \text{s.t.} & F_{\mathrm{x}}=\sum_{i=1}^{N}f_{\mathrm{x}i} \\ & F_{\mathrm{y}}=\sum_{i=1}^{N}f_{\mathrm{y}i} \\ & T_{\mathrm{z}}=\sum_{i=1}^{N}x_i f_{\mathrm{y}i}-\sum_{i=1}^{N}y_i f_{\mathrm{x}i} \end{array} \tag{1}$$

where forces F_{x} and F_{y} and z-torque T_{z} are the desired total forces to be applied to the sheet, $f_i=(f_{\mathrm{x}i}\ f_{\mathrm{y}i})$ are the allocated x and y forces for jet i, (x_i, y_i) is a jet's position, and $w_i=(w_{\mathrm{x}i}\ w_{\mathrm{y}i})$ are the weighting factors for each jet's contribution of x and y forces.

This problem can be decomposed recursively into sets of (smaller) optimization problems to be solved by subgroups at the next-lower levels of the hierarchy. Except for the bottom level, a "jet" i in Eq. (1) refers to the "virtual jet" that arises out of a group of jets. We are free to choose how to aggregate jets into virtual jets. One obvious heuristic is to follow the geometric layout, e.g., each tile of jets becomes a virtual jet (Fig. 4a). The advantage of this heuristic is that a modular structure of the system leads to a corresponding modular structure of the allocation algorithm.

Another heuristic is to aggregate jets in neighborhoods of a different parameter space, namely that of force directions and location along just one of the axes. This heuristic is inspired by the linear superposition model in the constraints of Eq. (1). We observe, for example, that all x-jets with the same y position are interchangeable with respect to the constraints in Eq. (1). In fact, if our system consisted only of N x-jets with the same y position, the instantiation of the allocation functions would simply be

$$f_{\mathrm{x}i}=\frac{F_{\mathrm{x}}}{N},\, f_{\mathrm{y}i}=0 \ \ (i=1,\ldots,N) \tag{2}$$

For discrete actuators, one can simply calculate the number of necessary jets as $F_{\mathrm{x}}/f_{\max}$ rounded to the nearest integer and then open that many jets ($f_{\max}$ is the maximum force available from a single jet).

Generally, rows of x-jets with the same y position and columns of y-jets with the same x position lead to particularly simple allocation functions within a row or column. Therefore, we have implemented an aggregation of jets where the N virtual jets are divided into "x modules," each with only x-directed jets with a common y position y_i, and "y modules," each with only y-directed jets with a common x position x_i (Fig. 4b). Now, allocation to these virtual jets has to compute only either $f_{\mathrm{x}i}$ or $f_{\mathrm{y}i}$ for the x and y modules, respectively, and allocation within a virtual jet follows Eq. (2) for x modules and its equivalent for y modules, instead of solving the more complex Eq. (1). Thus, the decomposition at one level greatly simplifies the allocation at another level.

Nested-loop partitioning

In a second approach to decomposition, we are developing a *nested-loop partitioning* approach that takes as input a nested-loop formulation of the task, as well as models of the system and the application (e.g., computing and memory limits, performance objectives) and a range of suggested hierarchical structures. In a nested-loop formulation, each vertex of the iteration space represents a given node or agent (e.g., sensor or actuator) (Ancourt *et al.* 1997). Examples are sensors and actuators embedded on a two-dimensional surface, such as the sensors and jets in the paper mover and the vibration sensors and actuators embedded on a surface grid for vibration control. There, a control cycle can be modeled as nested, two-dimensional loops iterating over these elements for the sensing, control, and actuation tasks. Partitioning groups and effectively parallelizes these loops such that the resource constraints are satisfied. For hierarchical applications, partitioning is applied recursively in order to determine an appropriate hierarchical structure and parameters, such as the branching factor and height of the hierarchy.

Adaptive Control of Problem Solving

The second element of our approach is the on-line tuning of solvers for the environment and problems of embedded applications. In general, the design of a problem solver for a particular problem depends on the problem type, the system resources, and the application requirements, as well as the specific problem instance. One approach to this design issue is the use of *adaptive control of problem solving*. We define adaptive control of solving as modifying the solver or the problem representation in response to environment or problem changes. Adaptive solving control makes use of a model or set of rules concerning the relationship between various problem, system, and application parameters and choices for solvers, heuristics, and problem transformations. These choices define the control parameters of the solver. This model or rule base also enables the prediction of the behavior of the solver defined by these control parameters. The predicted behavior can be used to monitor the actual on-line behavior of the solver and update the control parameters accordingly. If the predicted behavior is consistently at odds with the actual behavior, the accumulated performance feedback can be used to update the model or rule base itself.

A framework for adaptive solver control

These ideas are directly analogous to on-line adaptation in control systems (Åström & Wittenmark 1995; Levine 1996),

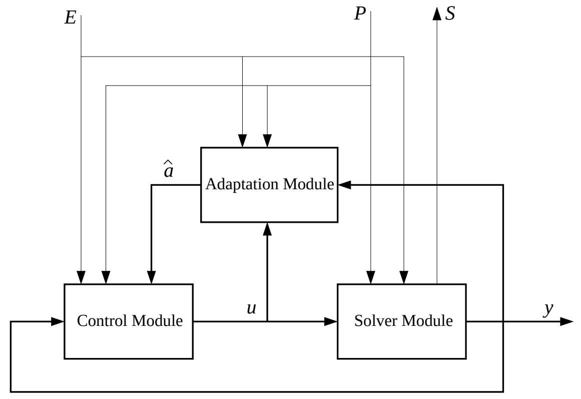


Figure 5: Adaptive solving system framework

with the solver taking the place of the plant. Based on this viewpoint, we propose a generic framework for the adaptive control of solving (Crawford *et al.* 2001), shown in Fig. 5. This framework includes three levels of increasingly sophisticated control: open-loop, feedback, and adaptive control. In a basic control system, the controller takes the reference input and outputs a control signal to the solver without any feedback. For such a feed-forward, open-loop control system to produce the desired behavior, the controller must be designed and tuned off-line using a very accurate model of the solver. Any inaccuracies in the model or variances to the system (e.g., the problems to be solved) will not be taken into account at run-time.

When the feedback loop is added to the control system, the controller gains the ability to react to the quality of the solver's output. The controller still must be designed and tuned off-line, but the system has a much better chance of performing as desired.

In an adaptive system, the controller is able to adapt to compensate for systematic performance errors caused by inaccurate solver modeling. The controller is designed and tuned off-line, but can be formulated parametrically in terms of solver parameters that may be unknown or imprecisely known. The model becomes more accurate over time and can also adapt to changes in the problems and solver's environment.

This functional framework is useful for structuring a principled discussion of adaptive solving approaches. The following description outlines the proposed functionality. Examples of possible implementations and instantiations of its elements are provided in (Crawford *et al.* 2001).

Referring to Fig. 5, the Solver Module represents software for problem transformation and solving, possibly containing multiple algorithms and heuristics among which to select, as well as tunable parameters. (By problem transformation, we mean changes to the problem formulation, such as changing the granularity of variable domains to improve efficiency.) The algorithms and parameters are selected with the input u, provided by the Control Module. This module makes the solver algorithm selection and tuning decisions based on the environment specification, E, the problem to be solved, P, the on-line (during a run) behavior of the solver, y, and some internal tuning parameters, a. The environment, E, includes models of both the system specifications and the application requirements. Including the on-line behavior of the solver as an input to the Control Module allows the solver control to have a feedback component. The role of the Adaptation Module is to monitor the behavior y of the Control Module/Solver Module system and adapt the parameters, a, of the Control Module to improve performance. Its outputs, $\hat{a}$, are the current estimates of the optimal Control Module parameters. The Adaptation Module would typically act at a much slower time scale than the feedback loop. As it actually modifies parameters of the Control Module, its actions would generally not be based on system behavior for a single problem or subproblem, as the feedback control would.

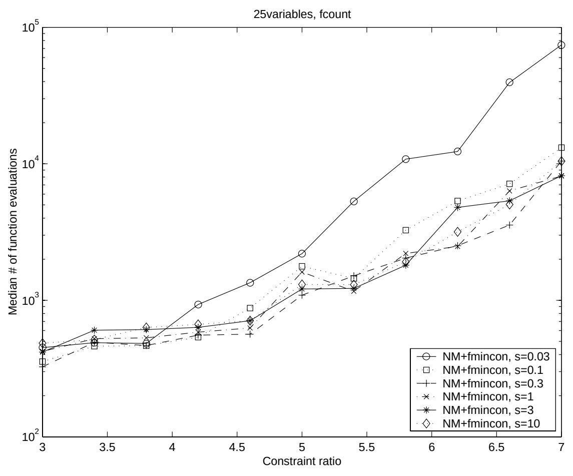
Both the Control Module and the Adaptation Module contain, either explicitly or implicitly, information about the solver and its expected behavior. A predictive model of expected solver behavior would provide a basis for the Control Module to make its solver parameter choices. The Adaptation Module would use the predicted behavior as a yardstick for measuring the true behavior, y. The controller parameters a adjusted by the Adaptation Module might then be parameters of the solver model.

Previous approaches to adaptive solving for different classes of problems have involved the first of the levels of control enumerated above and sometimes the second. In contrast, most of the work on adaptive solving to date has in fact not been adaptive, at least not in the control sense (Crawford *et al.* 2001). In other words, these approaches are not adaptive to the run-time environment of the solver, a crucial prerequisite for deploying solvers in embedded applications.

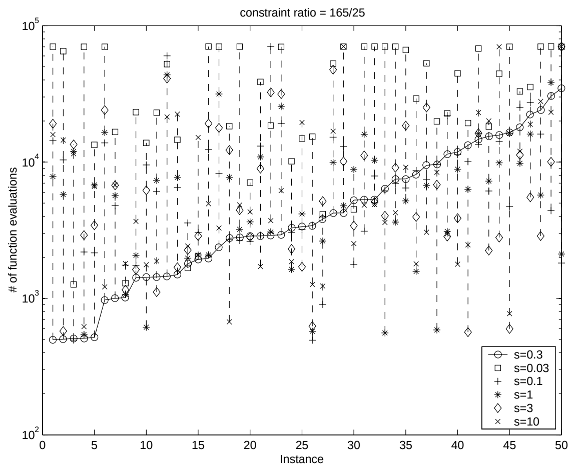
An example for feedback and adaptation in solving

We present the following example in order to illustrate the value of feedback and adaptation in on-line solving. It has been shown that the combination of global and local solvers can be particularly effective for complex, realistic problems (Shang *et al.* 2001b). Consider, for instance, a cooperative global/local solver for continuous constraint satisfaction problems that uses the Nelder-Mead algorithm (Lagarias *et al.* 1998) for global search followed by sequential quadratic programming for local search (fminsearch and fmincon, respectively, in the Matlab Optimization Toolbox (Matlab Optimization Toolbox 2001)). Such a solver has several tuning parameters that need to be set. One of these is the size of the initial simplex for the Nelder-Mead algorithm, which determines the search scale. An important indicator of complexity for this type of problem seems to be the constraint-to-variable ratio (constraint ratio for short). Thus, a first step might be to analyze the performance of various initial simplex sizes for various constraint ratios, as is shown in Fig. 6a for a 25-variable problem. Such an analysis provides the basis for a solver model and as such the foundation for an open-loop control scheme. This scheme specifies that the simplex size should be chosen as the one yielding the lowest median complexity given the problem's constraint ratio.

If, for instance, for a problem with constraint ratio 6.6, the simplex size were chosen based on Fig. 6a, a size of 0.3



a)



b)

Figure 6: Continuous constraint satisfaction problems with different initial simplex sizes: This figure shows data on sum-of-sines type continuous constraint satisfaction problems with 25 variables. The top diagram shows the median (over 50 instances) of the number of function evaluations required for problems with different constraint ratios, using initial simplex sizes s between 0.03 and 10. The bottom diagram shows, for a single constraint ratio, the variation among all instances and all simplex sizes

would be selected (the lowest point in the second-to-last set of data). Since this is the best choice only on average for this particular constraint ratio, it will not be the best choice for every problem instance. Fig. 6b shows the complexity for 50 different instances for this constraint ratio. The instances have been ordered by the complexity for the open-loop simplex size, 0.3. Clearly, there are a number of instances where different choices of the simplex would yield much better results. Therefore, a feedback law would be useful here in order to fine tune the system for the particular instance. When monitoring the on-line solver behavior, if the number of function evaluations were growing too large, the simplex size could be changed. This type of feedback could improve the solving time in cases like those on the right-hand side of Fig. 6b.

Finally, if, over the course of many problems, the initial open-loop estimate of the best simplex size is in error more often than not, the adaptation portion of the solver control comes into play. The Adaptation Module can monitor the behavior of the solver and compare it to that predicted by the solver model (here, the median complexity given the constraint ratio and the choice of simplex size). If this prediction appears to be incorrect, such as if the simplex size of 0.3 is yielding consistently worse results (and the feedback law described above has to act frequently), adaptation can adjust the solver model accordingly. Therefore, it may happen that, as more examples are gathered on-line, the estimate of the complexity for the 0.3 solver increases enough that 0.3 is no longer the preferred simplex size for problems with constraint ratio 6.6.

Thus, open-loop control implements parameter choices based on a solver model or a set of rules, either of which can be generated off-line based on analysis or statistical sampling. The feedback control component performs on-line corrections to compensate for individual instances deviating from the norm. Finally, the adaptive control component serves to correct persistent errors in the solver model or rule base, which can affect both the open-loop and feedback components.

Conclusion

Constrained optimization is at the core of many embedded applications. We are proposing a new framework for an (on-line) adaptive constrained optimization service for such applications that addresses two central issues: problem distribution and representation as well as adaptive algorithm selection and tuning.

The adaptation framework presented here allows for, as far as we are aware, the first principled classification of different techniques for adaptive control of problem solving. Our current focus is on generating suitable problem ensembles for complexity analysis, on a better understanding of the relevant problem parameters that control or at least predict complexity, and on cooperative solvers (Shang *et al.* 2001a; Fromherz *et al.* 2001).

We believe that this work is relevant for a wide variety of applications and is particularly important for large-scale, dynamic, embedded problems.

Acknowledgment This work is supported in part by DARPA under contract F33615-01-C-1904.

References

Ancourt, C.; Barthou, D.; Guettier, C.; Irigoin, F.; Jeannet, B.; Jourdan, J.; and Mattioli, J. 1997. Automatic mapping of signal processing applications onto parallel computers. In *Proc. ASAP 97*.

Åström, K. J., and Wittenmark, B. 1995. *Adaptive Control*. Addison-Wesley.

Bondarenko, A. S.; Bortz, D. M.; and Moré, J. J. 1998. COPS: Large-scale nonlinearly constrained optimization problems. Technical Memorandum ANL/MCS-TM-237, Argonne National Laboratory, Argonne, Illinois.

Chase, J. G.; Yim, M. H.; and Berlin, A. A. 1999. Optimal stabilization of column buckling. *ASCE Journal of Engineering Mechanics* 125(9):987–993.

Crawford, L. S.; Fromherz, M. P. J.; Guettier, C.; and Shang, Y. 2001. A framework for on-line adaptive control of problem solving. In *Proc. CP'01 Workshop on On-line Combinatorial Problem Solving and Constraint Programming*.

Fromherz, M. P. J., and Jackson, W. B. 2001. Force allocation in a large-scale active surface. *Submitted to IEEE Trans. on Control Systems Technology*.

Fromherz, M. P. J.; Hogg, T.; Shang, Y.; and Jackson, W. B. 2001. Modular robot control and continuous constraint satisfaction. In *Proc. IJCAI-01 Workshop on Modelling and Solving Problems with Constraints*, 47–56.

Fromherz, M. P. J.; Hoeberechts, M.; and Jackson, W. B. 1999. Towards constraint-based actuation allocation for hyper-redundant manipulators. In *CP'99 Workshop on Constraints in Control (CC'99)*. See also http://www.parc.xerox.com/fromherz.

Hogg, T., and Huberman, B. A. 1998. Controlling smart matter. *Smart Materials and Structures* 7:R1–R14.

Jackson, W. B.; Fromherz, M. P. J.; Biegelsen, D. K.; Reich, J.; and Goldberg, D. 2001. Control of real time large-scale systems: Airjet object movement system. In *Proc. Fortieth Conference on Decision and Control*.

Lagarias, J. C.; Reeds, J. A.; Wright, M. H.; and Wright, P. E. 1998. Convergence properties of the Nelder-Mead simplex algorithm in low dimensions. *SIAM Journal on Optimization* 9:112–147.

Levine, W. S., ed. 1996. *The Control Handbook*. CRC Press.

2001. *Matlab Optimization Toolbox*. The MathWorks, Inc.

Shang, Y.; Fromherz, M. P. J.; Hogg, T.; and Jackson, W. B. 2001a. Complexity of continuous, 3-SAT-like constraint satisfaction problems. In *Proc. IJCAI-01 Workshop on Stochastic Search Algorithms*, 49–54.

Shang, Y.; Wan, Y.; Fromherz, M. P. J.; and Crawford, L. S. 2001b. Toward adaptive cooperation between global and local solvers for continuous constraint problems. In *Proc. CP'01 Workshop on Cooperative Solvers in Constraint Programming*.

Wang, C., and Chang, F.-K. 2000. Diagnosis of impact damage in composite structures with built-in piezoelectrics network. In *Proc. SPIE, Smart Structures and Materials, v3990*, 13–19.

Yim, M. 1994. Locomotion with a unit-modular reconfigurable robot. Ph.D. Thesis, Dept. of Mechanical Engineering, Stanford University.