Controlling Asynchronous Team Design Environments with Simulated Annealing

Rich Quadrel, Robert Woodbury, Steven Fenves, Sarosh Talukdar

EDRC 48-29-92

Controlling Asynchronous Team Design Environments by Simulated Annealing

Richard W. Quadrel
Battelle Pacific Northwest Laboratories

Robert F. Woodbury
Department of Architecture
Carnegie Mellon University

Steven J. Fenves
Department of Civil Engineering
Carnegie Mellon University

Sarosh N. Talukdar
Department of Electrical Engineering
Carnegie Mellon University

The work reported in this paper was partially supported by the Engineering Design Research Center, a National Science Foundation funded center.

Send correspondence to:

Robert Woodbury
Department of Architecture
Carnegie Mellon University
Pittsburgh, PA, 15213, USA

FAX: +(412) 268 7819

email: rw@globe.edrc.cmu.edu

Short title:

Controlling A-Team Design Environments by Simulated Annealing

The work reported in this paper was partially supported by the Engineering Design Research Center, a National Science Foundation funded center.

Controlling Asynchronous Team Design Environments with Simulated Annealing

Abstract: *An organizational strategy for design environments,* asynchronous teams, *is reviewed. Simulated annealing is used to implement the necessary* contracting search *behavior of asynchronous teams. An example of an asynchronous team design environment controlled by simulated annealing is given from the building design domain. The simulated annealing algorithm used, which has been modified for distributed use and multi-criteria, non-preference objectives, is described.*

Recently *asynchronous teams (A-teams),* a model for the organizational structure of design environments, have been proposed and partially explored by Talukdar et al.[Talukdar 91]. A-teams stand on the conjecture that more appropriate organizational models for computer-based design environments might be found by organizing simple-minded computer tools into cooperative teams. A brief introduction to A-teams is given below; comparisons of A-teams with other concepts in the distributed problem solving literature may be found in [Quadrel 91] and [Talukdar 91].

1. Asynchronous teams (A-teams)

An asynchronous team is a distributed problem solving system that is organized as a hetrarchical network of autonomous, asynchronous agents. Informally, an A-team is characterized by five attributes:

Autonomous agents. Agents decide for themselves when to activate and how to contribute to the design process. The agents are sensitive to events in the environment at large, and activate primarily through stimulus-response conditions. But they are also imbued with decision-making capability to decide their behavior based on the available data, current state of the system and their own previous actions.

Broadcast communications. Information is broadcast throughout the environment as a whole. Agents are fully capable of deciding how to respond to generally broadcast messages. Broadcast communications have two advantages. First, they increase the flexibility of the system: the environment is easily reconfigured, since point-to-point communication needs no re-routing. Second, broadcast communications can improve feedback and iteration: the developer of the design environment doesn't need to specify pre-determined feedback points. Communication can originate from any agent and be accepted by any other agent.

Network structure. A-teams have no managers. There are no agents that explicitly activate other agents, although it is conceivable that agents change the state of the environment, causing other agents to self-activate. The absence of managers and controllers is a significant point of departure from the (human-based) organizations of other design environments. It is no longer necessary to attempt building highly-intelligent agents capable of control operations. Furthermore, it eliminates the need to constantly update computer-based managers as the configuration of the design environment changes.

Dynamic planning. A-teams do not impose a fixed or pre-specified sequence of agent activation. Since agents are primarily data-driven, their activation occurs opportunistically, resulting in an apparently dynamically planned behavior. Dynamic planning is better suited to handling unexpected contingencies, since static planners fail if an event occurs outside of their expectations.

Concurrent/asynchronous execution. The activation of agents is concurrent and asynchronous. Agents are self-activating, and many agents can work on the same problem simultaneously. In some implementations, the population of agents varies with the pending tasks: as more data is generated, agents spawn "clones" of themselves to handle the excess load. A-teams have no structural control bottlenecks, because solution generation, evaluation and modification occur in parallel; bottlenecks in A-team organizations are epiphenomena of system organization and problem type.

Without central control or a predefined plan of action, the key questions become, "How do asynchronous teams coordinate their efforts? What prevents individual agents from pursuing their own goals at the expense of other agents?" The answer lies in a key performance feature of A-teams: information found while searching one part of the design space can be used to influence concurrent search in another. When individual agents broadcast their successes and failures, other agents respond to these reports by altering their own behavior. Some agents may decide to abandon what now appears to be a fruitless exploration, while others decide to commit their resources to deeper dives into a promising region of the design search space. The search for satisficing solutions begins with great breadth, but eventually "contracts" around regions in the space that hold promising solutions.

2. A-teams and Design Search

Architectural design can be characterized as a search[1] for one or more satisficing solutions in the space of all possible design solutions [Woodbury 91]. The *design space* can be represented as a graph whose nodes are *solution instances* and whose arcs are *design transformations* that describe a path from one candidate solution to another. There is a corresponding *performance space* to which nodes in the design space may be mapped. A node in the performance space represents an evaluation of its corresponding solution in the design space. In either space nodes have *neighborhoods*. In design space the δ-neighborhood of node is the set of those nodes that can be reached by δ or fewer design transformations (and these transformations are the only means of creating or altering designs). In performance space the δ-neighborhood of a node is the set of those nodes that are less than δ distant from the node (according to the metric of the performance space). The δ-neighborhoods of a design taken from design and performance spaces do not, in

---

[1] *We use search here as a metaphor for moving through a space of possibilities. The distinctions of the search vs. exploration debate, that is, whether the search is guided by fixed or changing goals, the space is defined by a fixed or changing set of operators, or the process is reflexive, are immaterial to this characterization.*

general, contain the same designs. Small changes in a design may imply large changes in performance. Likewise, different designs may perform in very similar ways.

A graph representation of the design space is useful for seeing the progression of a design process. Design explorations can be followed by tracing through branches of the graph and examining the effects of modifications on solutions. The performance space is also useful for tracing the progression of design solutions, particularly the evaluations of those solutions. The performance space can typically be understood as an N-dimensional Euclidean space, where N is the number of performance variables. In this view, the distance between performance nodes gives some indication the relative value of solutions.

In building design problems, the design space is combinatorially explosive. Exhaustive exploration of the space is impractical, whether design is done by humans or computers. As a result, heuristics are used to expedite the process of finding good solutions. Heuristics commonly used by human designers often rely on reuse of previously found solutions, hierarchical decomposition and iterative refinement. For example, humans often design by first choosing a solution (node) in the design space that satisfies most of the goals and constraints of the problem as it is currently defined. Perhaps this solution is reminiscent of a solution from a previous project or perhaps it is simply the designer's "best guess" as a starting place for exploration. From here, the designer applies a number of transformations (arc-traversals) to the initial guess until satisfactory solutions are encountered.

Computers, having different processing capabilities, have been programmed to use different search methods. Specifically, with computers, large stocks of states may be maintained and spaces may be systematically traversed. The well-known *weak methods* provide formal but combinatorial algorithms for searching spaces. Numerous heuristic methods have also long been employed. More recently parallel computation techniques have been used both to implement the methods above and to introduce new methods of search. Asynchronous teams present one approach to parallelism. In an A-team, multiple nodes are explored simultaneously by teams of agents. Their findings are broadcast to all team members, who may redirect their efforts based on the success or failure of their co-workers. In this sense, search in one portion of the design space

influences search elsewhere. As the search matures, the net effect is that unfruitful explorations are abandoned in favor of search in more promising directions, and the agents tend to work in the same parts of the search space. We call this phenomenon of a multi-agent system *contracting search.*

3. Simulated Annealing

Simulated annealing is a variant of a heuristic called *iterative improvement* (also known as the *steepest descent method*). In iterative improvement, a (random) solution is initially generated and its cost is calculated. One attempts to improve this solution by "perturbing" or modifying it in some way. If the cost of the new solution is less expensive than the first, then the new solution is accepted, otherwise it is rejected. This process continues until no further solutions are accepted, in which case the procedure has found a local optimal solution (in terms of cost). Simulated annealing also improves solutions iteratively, but uses probability to accept some "uphill" moves, allowing it to escape from local minima. The notion of simulated annealing originated in a paper by Metropolis et al.[Metropolis 53], and was more widely developed in the 1980's beginning with a paper by Kirkpatrick [Kirkpatrick 83]. A relatively current overview and comparison with other techniques is given by Rutenbar [Rutenbar 89].

Physical annealing finds an atomic arrangement that minimizes the amount of energy in a substance. Simulated annealing plays on a metaphor to physical annealing to solve optimization problems. Metropolis et al. describe this technique as applied to physical systems:

> *Each state of a physical system has some unique energy E. Simulating the behavior of the system then becomes a question of displacing a single particle in it at each step; if the configuration that results from such a move has an energy lower than E (i.e., $\Delta E \leq 0$), then the move is accepted and further perturbations are carried out from the new configuration. If, however, the change in energy ΔE is greater than zero, then the move is accepted with some probability $P=e^{-\Delta E/kT}$, where T is the temperature of the system and k is Boltzmann's constant (1.38062×10^{-23} j/°K)." [Metropolis 53]*

Using a landscape metaphor, one can describe the problem as follows: you are standing on a hilly plot of land, and you wish to find your way to the bottom of the lowest valley. Unfortunately, it is very foggy and you cannot see where the lowest valley is. If you take a step downhill, you know that you are moving in the right direction - this is analogous to an "accepted" perturbation in simulated annealing. Eventually you will reach the bottom of some valley, although you do not know if it is the lowest valley. In this case, you might take some uphill steps to extricate yourself from this valley, hoping that by first going higher, you will eventually find a path that takes you to a lower valley. In simulated annealing, this is analogous to "acceptance with probability." The likelihood that you will take uphill steps decreases, however, depending on how far you have already walked: it is much more likely early in your trek than later.

Simulated annealing presents one possible method of implementing contracting search. This technique moves the search from a "high temperature" state to a "low temperature" state, imitating the physical annealing process for transforming liquid materials to solids. One of the key features of simulated annealing is that the character of the search changes as the temperature decreases. At high temperatures, nearly all search explorations are "accepted," meaning that most solution nodes are eligible for modification. The system behaves somewhat chaotically here, indiscriminately operating on most solutions, both good and bad. As the temperature is lowered, however, an increasing number of solutions are "rejected" if they don't perform as well as their predecessors. Agents then dedicate their attention to the most promising solutions, abandoning those that appear to be fruitless.

In this paper we report an application of simulated annealing as a control strategy for an asynchronous design environment. We use simulated annealing for three reasons: (1) the complexity of building design demands heuristic, rather than exhaustive search; (2) the operators used in building design are non-monotonic in the designs they generate, thus making search that is formally based on pruning strategies (e.g., the A* algorithm) problematic; and (3) the effect of the simulated annealing temperature schedule appears to closely correspond to the contracting search behavior of asynchronous teams. However, the classical formulations of simulated annealing are suited to neither asynchronous nor

design environments. They maintain only a single open search path, so cannot be effectively used by multiple agents in an asynchronous environment. They use a single measure of solution cost, so do not model well the realities of design, where multiple incommensurable criteria are always at play. To respond to these issues, we have modified the simulated annealing algorithm so that it maintains in the generation process not a single design at a time, but a set (called the *accepted set*) of designs. Any design in the accepted set is a candidate for perturbation by the algorithm. A subset (called the *non-dominated set* or the *ND set*) of the accepted set contains those designs found so far that are not unambiguously worse than other candidate designs (i.e., they are mutually non-Pareto-dominant). We give the details of Pareto-dominance and of our algorithm in Appendix A.

4. An example application

Anarchy [Quadrel 91] is a working prototype of an asynchronous design environment. Its 20 agents are capable of generating, evaluating and modifying designs of medium- and high-rise office buildings. The construction of the agents allows them to activate and execute autonomously, concurrently and asynchronously. The network organization of agents, combined with distributed data stores and broadcast communication, makes the environment flexible: components can be inserted or deleted with virtually no modifications to other system components. Through its control via a simulated annealing algorithm the environment's general behavior can be described as contracting search: the search commences with an emphasis on breadth, then narrows its focus to explorations of only the most promising solutions.

At the core of each agent in Anarchy is a computer program that accepts input, does computation and produces output. Generator agents (9 in number) accept high-level descriptions (e.g., specifications) and produce low-level descriptions (e.g., designs). Evaluators (3 in number) accept low-level descriptions (e.g., designs) and produce high-level descriptions (e.g., measures of performance). Modifiers (3 in number) accept and produce descriptions at the same level of abstraction. The remaining agents (5 in number) handle version control, the annealing schedule and human-computer interface. The exact nature of the computer programs used is unimportant to the simulated annealing method.

For example, several of the generator tools in Anarchy were written for the Integrated Building Design Environment [Fenves 92] and were used directly from that implementation. Other tools are instances of a *design-by-selection* method, and are programs written in the EDIC shell [Quadrel 91], itself a variant of the earlier EDESYN [Maher 87].

The agents and the data sources (*aspects*) from which they accept input and to which they write output are organized in a network. The organization of Anarchy that was used in the example discussed here is partially diagrammed in Figure 1 (which, for the sake of readability, shows the generator agents only). In this figure ovals denote agents and rectangles denote *aspect classes* or templates that describe the data-structure of the aspects actually used by agents.

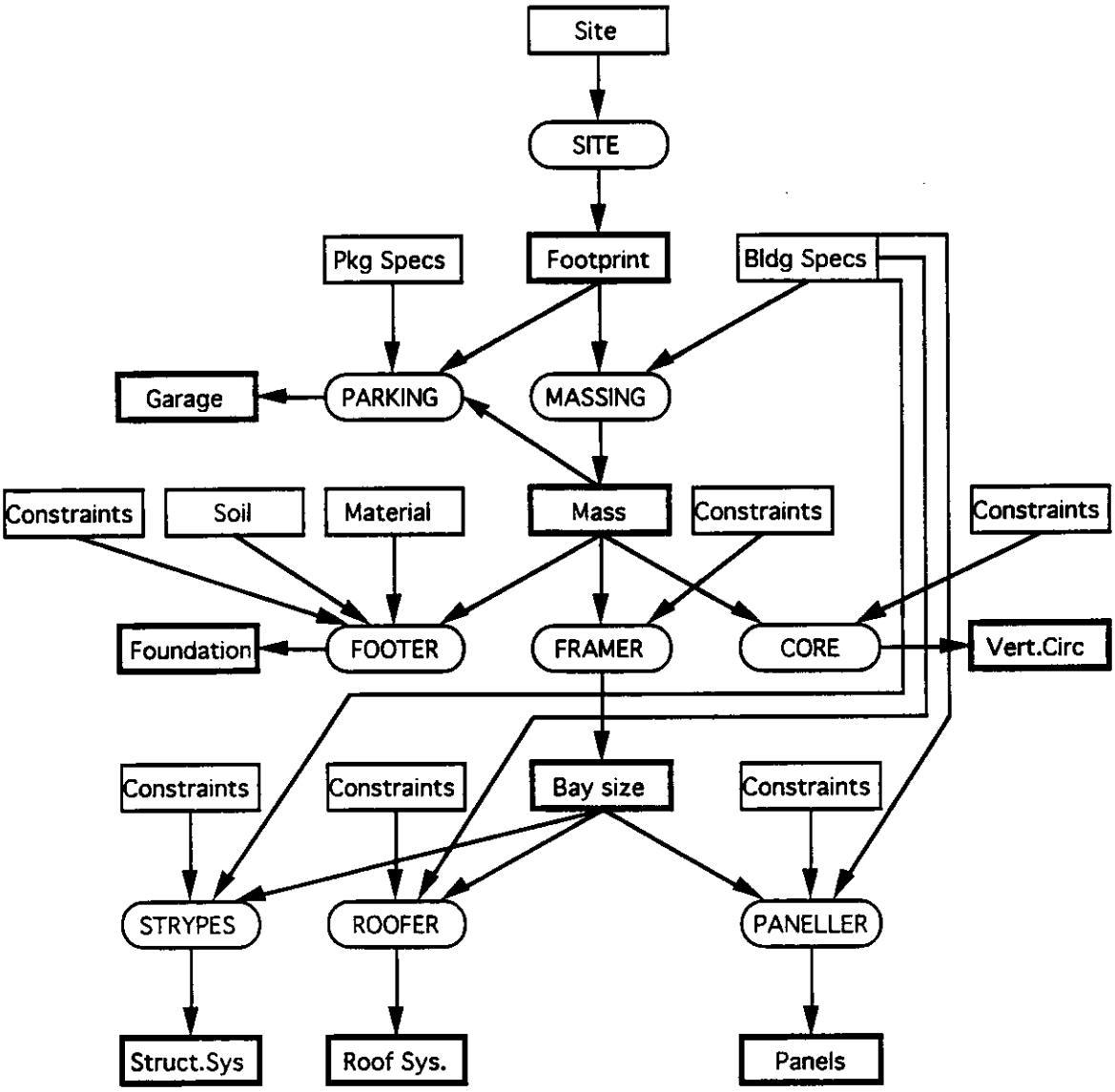


Figure 1: Organization of generator agents and their aspects classes in Anarchy.

Agents in Anarchy are activated in a design system by the action of *modules* that control the execution of individual agents and perform the roles of self-activation and modification of the data environment in the system. Each type of agent (e.g., generator, evaluator, modifier) has its own module structure; individual modules are instances of this structure.

Module structures in turn are constructed as sequences of building blocks from the set {*retriever, facilitator, deliverer, listener, executor, talker*}. A more complete description of the Anarchy architecture, including its module structure, version control system and interface, may be found in [Quadrel 91].

Design Problem Description

Anarchy was presented with a multi-story office design problem and produced 45 solutions in response. Five of these solutions are non-dominated, meaning that in terms of performance evaluation, none of these solutions is dominated by any other. Given that design solutions are evaluated by multiple criteria, these five solutions represent Anarchy's best estimate of the Pareto surface.

SITE		**BUILDING PROGRAM**	
Location	Pittsburgh, PA	Functions	office/commercial
Latitude	40.5° N	Gross req. area	50,000 sf.
Longitude	79.9° W		
EW dimension	160 ft	**CLIMATE**	
NS dimension	230 ft	Ave. Dec. temp	25°
Land cost	$15/sf	Ave. Jun. temp	80°
Setbacks	20, 10, 20, 10 ft	Dec. solar rad.	(0, 33, 66, 33)
Max. stories	12	Jun. solar rad.	(0, 20, 48, 20)
SOIL		**MATERIALS**	
type	clay	Steel	40,000 psi
consistency	hard	Steel (yield)	50,000 psi
obstructions	sparse	Concrete	4000 psi
vibration sens	no	Timber	800 psi
bedrock depth	70 ft		
unit soil wt.	100		
pore pres.	1248		

Table 2: Input information for the design problem.

The building site is located on a flat 150'x80' lot in Pittsburgh. Soil conditions and weather conditions are typical for this area. The building program specifies a mixed-use office/commercial building of 50,000 square feet. In general, the input information

required by the environment is defined by the site, soil, building program, climate and material aspects. The information contained in these aspects is summarized in Table 2.

Methodology

Each agent is constructed with a listener module that awaits broadcast signals from other agents within Anarchy. When the aspects listed in Table 2 are "entered" into the system, appropriate signals are broadcast throughout the environment. Some of the agents find these signals interesting and begin execution, using these aspects as inputs. Others choose not to respond to these signals and return to a listening state, waiting for different aspects to be generated.

The output aspects created by agents will activate other agents, who will in turn use these for input. This sequence of events will result in all generative agents activating exactly once, producing the aspects necessary to describe the first solution, called Solution 0. Anarchy requires nine solution aspects to compose a complete building description: a building footprint, mass, frame, core, parking area, structural system, foundation, panel and roof system. Each of these aspects is inserted into its proper location in the 0th row of the solution table, as shown in Figure 3.

As aspects of Solution 0 come into existence, the solution is evaluated by the three evaluator agents: the thermal evaluator, lighting evaluator, and constructability evaluator. The evaluators need not wait for all of the solution aspects to be in place - they can begin their evaluations as soon as the appropriate partial solution is available. These evaluations are also inserted into their proper places in Row 0 of the solution table, as shown in Figure 3.

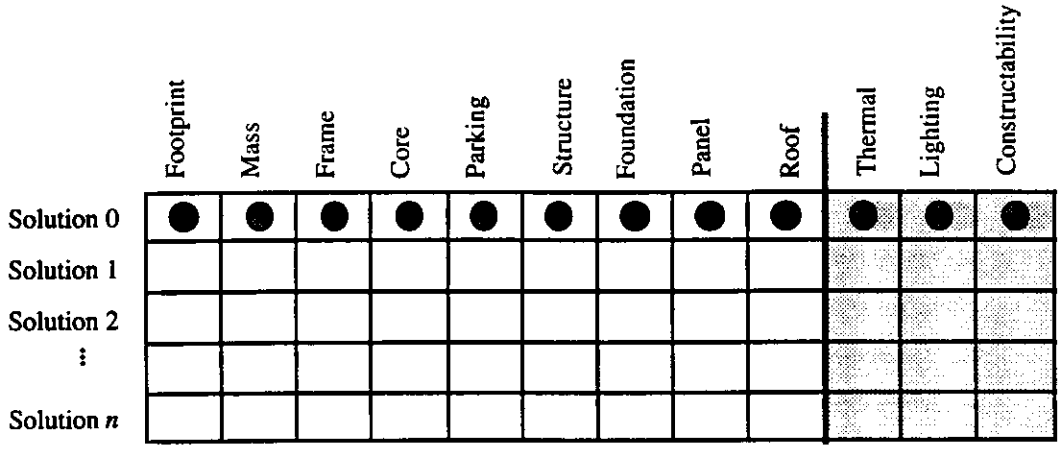


Figure 3: Row 0 of the Solution Table (after evaluations). A row must be completely filled before it can be sent to the Annealer.

Once all evaluations have been made, an agent called the Solution Tool sends Solution 0 to an agent called the Annealer. Since this is the very first building generated, the Annealer accepts Solution 0 without question. With all subsequent solutions, however, the Annealer performs steps **f1** and **f2** of the multi-criteria simulated annealing algorithm (see Appendix A). It also updates the target vector (a vector which is used by the distance function - see Appendix A) if any of the evaluations of the candidate solution are an improvement over the evaluations of the current target solution. In each step, a solution can only become a candidate when all of its evaluation aspects are present.

The acceptance of a candidate solution implies that it is eligible for modification. If an acceptance signal has been broadcast to the environment, the thermal, lighting and constructability modifiers inspect the evaluations to determine if the solution needs improvement. Each of these modifiers can make changes to the input aspects used by the generators. For example, the thermal modifier adds constraints that are likely to cause the generator to move circulation cores to the outside or to add an atrium to the building design. If the solution is rated "satisfactory" in a particular category, the corresponding modifier returns to its listening state. Otherwise, the modifier attempts to improve the solution's performance by providing feedback to generator agents. The process of

supplying feedback is called problem restructuring: the modifiers change constraints and input values, essentially preparing a new problem for the generators to solve. Instantiations of these new input aspects will automatically cause the generators to (re)activate and process the new information. The existence of multiple modifiers is sufficient to guarantee that multiple candidate solutions will be generated. Each modifier alters a problem description independently to produce a new problem description. The actions of modifiers are not combined in a single problem description.

The process of design generation, evaluation and modification continues until one of the following conditions becomes true:

- the annealing temperature of the environment has reached its "freezing point."
- the annealing temperature has decreased four times without any solutions being accepted.
- every accepted solution is either satisfactory or has been already subject to at least one modification.

At this point, the ND set (the solutions that remain in the accepted set and that are not dominated by other solutions in that set) are the "best" solutions found by the environment, and represent Anarchy's approximation of the Pareto surface for this problem.

The First Solutions

Anarchy was presented with a design problem as specified in Table 2. After all of its generators and evaluators activated once, Solution 0 (the first solution) was produced. This design is a rectangular eight-story office building whose dimensions are 60 feet in the east-west direction , approximately 105 feet in the north-south direction, and 120 feet high. The building has eight stories, with a floor-to-floor height of twelve feet. Each floor contains 4 EW bays and 7 NS bays. There are two vertical circulation cores, approximately 22 feet square and located near the center of the building. The solutions for the foundation elements, roof system, structural system and curtain wall panel selection are given in Tables 4 and 5.

FOUNDATION ELEMENT		ROOF SYSTEM	
Applied load	118.295 psf	Span	40 ft
Foundation type	spread footing	Live load	12 psf
Material	reinforced concrete	Available depth	12 in
Cross section	square	Low sound trans.	yes
Load resistance	end bearing	Unfinished clg.	yes
Construction	cast-in-place	High thermal cap.	yes
Installation method	excavated	Roof type	1-way rib. conc. slab
Footing width	5.44 ft	Slope	flat
Footing thickness	23 in	Service plenum	between ribs
Steel ratio	0.004	Ave. dead load	73.33 psf
		Fire protection	4 hrs

Table 4: Foundation and Roof system proposals for Solution 0.

STRUCTURAL SYSTEM		PANEL SYSTEM	
Occupancy	commercial	Dominance	horizontal
3D lateral system	orthogonal 2D	Frame	inserted
2D system	rigid frame	Facing	aluminum sheet
Material	steel	Insulation	fiberglass
2D horizontal	concrete	Fenestration	fixed
Support type	2 y edges	Joint type	lap
Subdivide type	none	Fastener location	horizontal
		Panel width	8 ft

Table 5: Structural and Panel systems proposals for Solution 0.

Table 7 shows how the evaluators rated Solution 0. Thermal ratings are given in terms of millions of Btuh's gained or lost during a typical winter and summer day. Lighting ratings are given on a scale from 1 (best) to 5 (worst). Constructability ratings are binary: either the solution PASSed or FAILed.

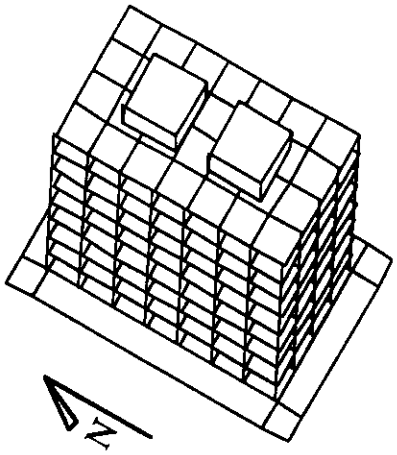


Figure 6: Framing plan and vertical circulation cores for Solution 0.

T	Winter day (MBtuh)	-0.028244
	Summer day (MBtuh)	2.53292
L	Direct lighting	5
	Natural lighting	4
C	Structural	PASS
	Dimensional	FAIL

T THERMAL

L LIGHTING

C CONSTRUCTABILITY

Table 7: Evaluations of Solution 0

As we might expect from a multi-story office building, thermal performance is particularly poor in the summer, when the internal gain generated by people, lights and equipment causes high cooling loads. In the winter, this internal gain offsets heat loss, tending to stabilize interior temperatures. Lighting conditions for this building have been rated as relatively poor: the narrow width admits little northern light, and the low perimeter-to-floor-area ratio suggests that this building requires significant artificial lighting. Since this building does not use "core" or "tube" framing systems, the constructability evaluator need not search for conflicts between the framing plan and core location, and consequently PASSes the candidate. The core dimensions, however do not align with the bay dimensions, and the candidate solution is FAILed by the constructability evaluator from a dimensional standpoint.

Solution 0 is automatically accepted, and is eligible for modification by all three modifier agents. Their efforts will produce three new solutions, each of which will be evaluated and sent to the annealer for acceptance or rejection. If accepted, these new solutions will also be eligible for modification, producing yet another generation of solutions.

The next generation

Solutions 1, 2 and 3 are the direct descendants of Solution 0: they are new solutions that have been modified based on thermal, lighting and structural compatibility, respectively.

Because of Solution 0's thermal performance, the thermal modifier "restructured" the design problem for the Circulation Tool, which in turn contributed a new design for Solution 1. This was done by adding a constraint to its input that prevented it from locating the service core within the building. The Circulation Tool responded by keeping two separate cores, but located them at the north and south ends of the building. The thermal modifier selected this strategy "knowing" that the external core on the south side would shade the building from summer solar radiation. This was a "radical" recommendation that would be more likely to be suggested early in the design process. As the search matures, the thermal modifier would instead tend to suggest small modifications to the building's proportions - a strategy that does not have as many far-reaching consequences.

The lighting modifier's suggestion was also relatively radical: it suggested that an atrium be added to the building. To do this, the lighting modifier removed a constraint on the input for the Framer Tool that prevented an atrium space to be allocated within the building. The resulting solution (Solution 2) had better lighting performance than its parent, Solution 0, but the new atrium decreased its thermal performance.

Solution 3 was the result of recommendations from the structural compatibility modifier. This is the only modifier that is capable of making direct changes to the solution itself. In this case, the compatibility modifier changed the dimensions of the vertical circulation cores so that they would align with the existing bay dimensions. Since this change has no effect on thermal or lighting performance, Solution 3 looks and performs exactly like Solution 0, except for the fact that Solution 3's cores have a slightly different shape, allowing it to PASS its structural compatibility evaluation.

Exploring the Search Space

Representation of the search space

The history of solution generation can be depicted as a directed, acyclic graph, in which the square nodes are solutions and arcs are actions that transform one solution into another (Figure 8). The root of the graph is Solution 0 and its children nodes constitute solution points in the design search space. Solutions that remained in the ND Set after completion of the experiment (the Pareto surface solutions) are shaded grey. Nodes with "Xs" indicate that the Annealer rejected this solution.

Arcs represent the transformation from one solution to another, which involves both the actions of the modifier (to restructure the problem) and the actions of the generators to produce a new solution. Truncated arcs indicate that the solution received a satisfactory evaluation, eliminating the need for that modifier to take action.

Search chronology

While Figure 8 shows the parent-child relationships between various solutions, it does not indicate the chronology of solution generation. The order in which solutions were generated is shown in the chronology diagram in Figure 9. Solutions in the same column were generated (roughly) concurrently. Those that appear above the horizontal line were accepted by the Annealer, those below it were rejected. Anarchy uses the Accept-Reject Count method for deciding when to decrease the temperature. The annealing temperature was decreased by 15% when either four solutions were accepted or eight were rejected. As the temperature T decreases, the probability that the Annealer will accept dominated solutions also decreases (according the formula $e^{-\Delta C/T}$). The results of the Anarchy experiment confirm this behavior: Figure 9 shows an increased number of rejections and a decreased number of acceptances as the temperature is lowered.

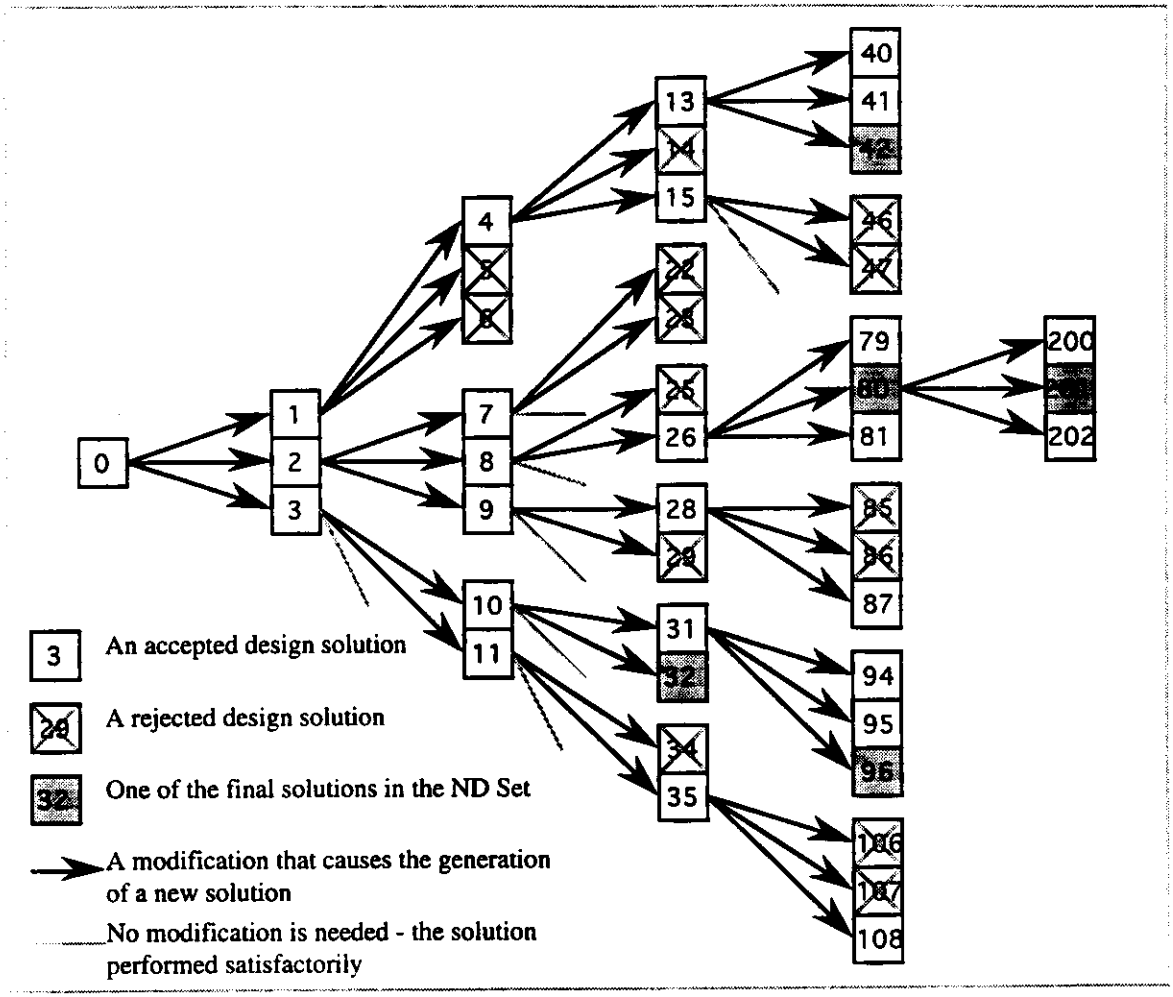


Figure 8: Exploration of the design search space. Numbers on the nodes are only labels, they do not represent any sequencing information.

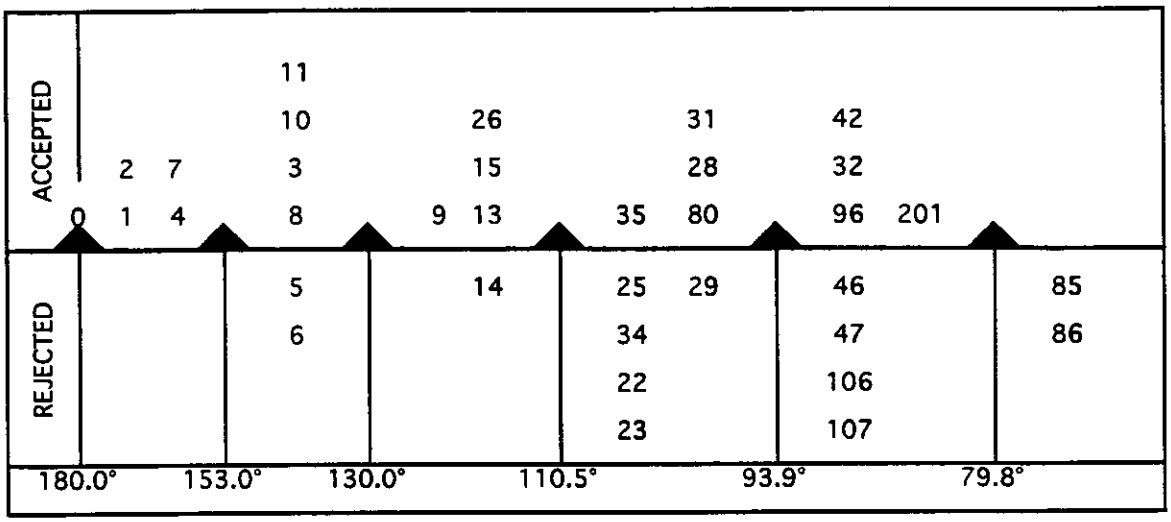


Figure 9: Chronology of solution generation.

Targets

Regardless of whether a solution has been accepted or rejected, the Solution Tool checks the solution's evaluation to determine if it surpasses any of the values on the target vector. The target vector contains the best possible evaluations in each category, regardless of what solution they belong to.

In Table 10, the columns **A...I** refer to state of the system at successive points of time (not necessarily spaced at equal intervals) in a run. In column A is shown that the target vector is initialized by the evaluation of Solution 0. Solution 1, however, immediately improves on Solution 0, as indicated by the improved thermal performance values in column B. Solution 2 improves on Solution 1 in the natural lighting category (column C). This continues until column I, where the best evaluations are found in a combination of Solutions 96, 32, 200, 7 and 0. There is no single building that can match the evaluations in column I: they are simply target values used by the Annealer to compute distances in the performance space. As the problem solving process matures, the targets improve.

	A		B		C		D		E		F	
Winter loss	-0.28	0	-0.07	1	-0.07	1	-0.07	1	-0.07	1	-0.07	1
Summer gain	2.533	0	2.517	1	2.517	1	2.517	1	2.517	80	2.517	31
Direct lighting	5	0	5	0	5	0	5	0	2	80	2	80
Natural lighting	4	0	4	0	2	7	1	7	1	7	1	7
Structural	PASS	0	PASS	0	PASS	0	PASS	0	PASS	0	PASS	0
Dimensional	FAIL	0	FAIL	0	FAIL	0	PASS	7	PASS	7	PASS	7

	G		H		I	
Winter loss	0	96	0	96	0	96
Summer gain	2.403	96	2.368	32	2.368	32
Direct lighting	2	80	2	80	1	203
Natural lighting	1	7	1	7	1	7
Structural	PASS	0	PASS	0	PASS	0
Dimensional	PASS	7	PASS	7	PASS	7

Table 10: Modification of the targets list.

The ND Set

The Non-Dominated (ND) Set contains Anarchy's best estimate of the Pareto surface at any given time. Solutions that appear in this set are those whose evaluations are not dominated by the evaluations of any other solution. Initially, Solution 0 is the only solution in the ND Set. New solutions can only join the set when they share a non-dominating relationship with the other solutions already in the ND Set. There are three possibilities that the Solution Tool must consider when a new solution is generated:

a. the new solution completely dominates a subset of solutions in the ND Set. In this case, the subset is removed from the ND Set and replaced by the new dominating solution.

b. the new solution does not dominate any of the solutions in the ND Set, nor do any solutions in the Set dominate it. In this case, the new solution is simply accepted and added to the set.

c. the new solution is completely dominated by a subset of the solutions in the ND Set. There is a probability that this new solution will be accepted by the Annealer, but it will never appear in the ND Set.

The ND Set changes during the course of problem solving. New solutions are added if they satisfy the non-dominated criterion, others are eliminated when they become dominated by better solutions. Table 11 shows the changes to the ND Set during the course of Anarchy's experiments.

Solutions in ND List	
0	**TEMPERATURE = 180** The ND list is initialized with Solution 0.
1	Solution 1 dominates 0, and replaces it.
1, 2, 4, 7	Solutions 2 (lighting), 4 (summer thermal) and 7 (lighting) are added.
	TEMPERATURE = 153
1, 2, 3, 4, 7, 8	Solutions 8 (dim. construct.) and 3 (summer thermal) are added.
1, 2, 4, 7, 8, 10	Solution 10 dominates 3, and replaces it.
1, 4, 7, 8, 10, 11	Solution 11 dominates 2, and replaces it.
	TEMPERATURE = 130
1, 4, 7, 8, 10, 11, 13	Solution 13 (lighting) is added.
1, 7, 10, 11, 13, 15	Solution 15 dominates 4 and 8, which are replaced.
	TEMPERATURE = 110.5
1, 7, 10, 11, 13, 15, 31, 80	Solutions 31 (summer thermal) and 80 (direct light) are added
	TEMPERATURE = 93.9
7, 11, 13, 80, 96	Solution 96 is very successful, and dominates 1, 10, 15 and 31
7, 11, 13, 32, 80, 96	Solution 32 (winter thermal) is added
32, 42, 80, 96	Solution 42 dominates 7, 11 and 13
32, 42, 80, 96, 201	Solution 201 (direct lighting) is added to the ND list

Table 11: Evolution of the ND Set during the course of problem-solving.

The Non-Dominated Solutions

The five remaining solutions in the ND Set (32, 42, 80, 96 and 201) represent Anarchy's approximation of the Pareto surface. They are, in other words, the "best" solutions

produced by the system. The following figures and tables give a description of each solution.

As shown in Figures 12 and 13, Solution 32 performed best with respect to heat gain during the summer, due to the shading benefits provided by the external core. Its compact shape limits heat loss during the summer, but is a liability for natural lighting. The constructability evaluator PASSed this building on both structural and dimensional categories.

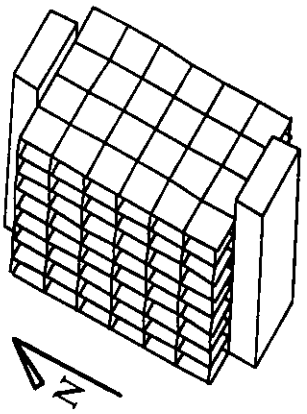


Figure 12: Framing plan and vertical circulation cores for Solution 32.

T	Winter day (MBtuh) Summer day (MBtuh)	0 2.36761
L	Direct lighting Natural lighting	5 5
C	Structural Dimensional	PASS PASS

T THERMAL

L LIGHTING

C CONSTRUCTABILITY

Table 13: Evaluations of Solution 32.

As shown in Figures 14 and 15, Solution 42's narrow proportions make it particularly good for natural lighting. Its "1" rating is the result of a very high perimeter-to-floor-area ratio.

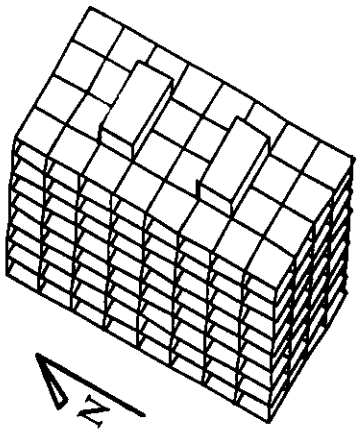


Figure 14: Framing plan and vertical circulation cores for Solution 42.

T	Winter day (MBtuh) Summer day (MBtuh)	-0.100127 2.52958
L	Direct lighting Natural lighting	5 1
C	Structural Dimensional	PASS PASS

T THERMAL
L LIGHTING
C CONSTRUCTABILITY

Table 15: Evaluations of Solution 42.

Solution 96, shown in Figures 16 and 17, is another narrow building, but unlike 42, its cores are external on the east and west sides. 96 pays the price in the form of a reduced lighting evaluation (compared to 42), but registers an improvement in both summer and winter thermal performance.

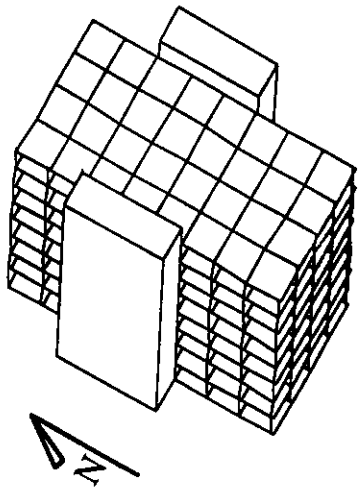


Figure 16: Framing plan and vertical circulation cores for Solution 96.

T	Winter day (MBtuh) Summer day (MBtuh)	0 2.40347
L	Direct lighting Natural lighting	4 3
C	Structural Dimensional	PASS PASS

T THERMAL
L LIGHTING
C CONSTRUCTABILITY

Table 17: Evaluations of Solution 96.

Solution 80 (Figures 18 and 19) and 201 (Figures 20 and 21) are similar buildings: both are elongated in the east-west direction, both contain single, central cores. Their only difference is the proportion of their lateral and longitudinal dimensions. Solution 80 is slightly "boxier," giving it a slight advantage in winter thermal properties while Solution 201 is slightly narrower, giving it slightly better northern lighting.

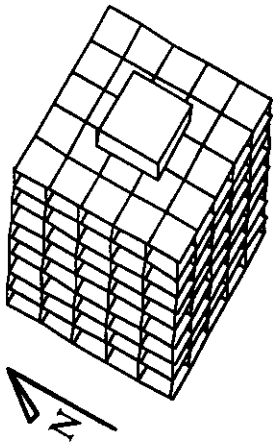
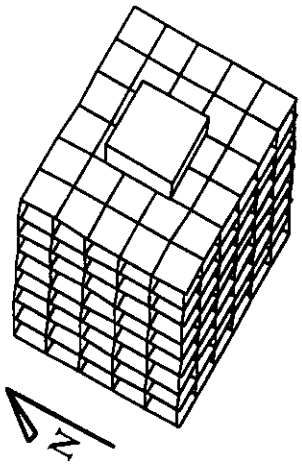


Figure 18: Framing plan and vertical circulation cores for Solution 80.

T	Winter day (MBtuh) Summer day (MBtuh)	-0.009074 2.44564
L	Direct lighting Natural lighting	2 5
C	Structural Dimensional	PASS FAIL

T THERMAL
L LIGHTING
C CONSTRUCTABILITY

Table 19: Evaluations of Solution 80.



T	Winter day (MBtuh) Summer day (MBtuh)	-0.019064 2.42869
L	Direct lighting Natural lighting	1 5
C	Structural Dimensional	PASS FAIL

T THERMAL
L LIGHTING
C CONSTRUCTABILITY

Figure 20: Framing plan and vertical circulation cores for Solution 201.

Table 21: Evaluations of Solution 201.

It is interesting to note that none of the buildings with atriums appeared in the final ND Set. According to Anarchy's evaluation, atriums are a thermal liability for which their lighting benefits can not compensate.

5. Observations and Conclusions

Experiments with the Anarchy system demonstrate that simulated annealing is a promising technique for implementing contracting search in an asynchronous design environment. The experiments show that the search through the design space was initially very broad (many solutions were accepted), then later "contracted" around a few of the best solutions (as temperature decreases and solutions were more likely to be rejected). The stochastic nature of the simulated annealing algorithm allowed Anarchy to find very good solutions as a result of accepting some bad ones. For example, Solution 1 remained in the ND Set for a relatively long period of time. Solution 4, a child of 1, was accepted by the annealer in spite of its poor performance. Modifications to Solution 4 produced Solutions 13 and 15, both of which were great improvements. Standard hill-climbing approaches would have rejected 4, thereby eliminating the possibility of ever reaching 13 or 15.

The designer has some controls by which the behavior of the asynchronous design environment can be altered. The most effective method is to change the annealing schedule. By slowing the pace of annealing, the environment can explore more solutions at a given temperature level, increasing its probability of finding an optimal solution (if there is one). On the other hand, accelerating the annealing process will allow designers to find solutions in a much shorter period of time, although these solutions may not be the best. In many cases, the most appropriate annealing schedule is found through experimentation, although there are some rules of thumb that may guide the designer.

It is a phenomenon of our simulated annealing algorithm that modifiers appear able to "use" the current annealing temperature to choose their modification strategies. When the

annealing temperature is high, the radical suggestion of the lighting modifier to add an atrium to the building to improve lighting performance likely would be accepted, even if it reduces performance at first. As the temperature decreases, only less drastic modifier recommendations, such as a slight modification to the building proportions, would be likely to be accepted. The net effect is that sweeping changes are more probable early in the design process, providing many different starting points for search. Later, as the search contracts, modifiers appear to provide "fine-tuning" adjustments to the solutions.

A method of control that amplifies the above effect is to change a modifier's strategies for selecting among modification options based on the current annealing temperature. When the temperature is relatively high, the modifiers tend to choose more "radical" recommendations than when it is low. If this policy were to be changed - that is, if recommendations that have extensive consequences were chosen later in the design process - it could theoretically have one of two effects. On one hand, the system may require more time to find a "stable" set of solutions, since such drastic changes are likely to require extended computation to determine downstream effects. On the other hand, one could also argue that these radical recommendations are more likely to be rejected because of the lower annealing temperature, leading to less, not more, computation. Which of these two scenarios would actually occur is still an open question, and requires further experimentation. One interesting possibility would be to change a modifier's strategy based on the success or failure of past recommendations. A modifier would need to keep a history of actions and their effects for a variety of problems, and it would require sufficient intelligence to match this history against the current problem. Research efforts in case-based reasoning and machine learning may provide some information on the possibility of guiding an agent's knowledge based on previous experience.

A final "tuning" method is to change the modifier's thresholds, essentially making it more or less "demanding" about the quality of the solution. Making such adjustments allows the designer to trade off the amount of time required to arrive at a set of final solutions against the overall quality of those solutions.

The present version of Anarchy provides no facilities for externally changing problem specifications and constraints. In principle, the asynchronous execution of the Anarchy

environment makes external intervention trivial: a human would be simply another agent, providing evaluations, solutions and modifications to problems at will. It remains to be seen what the practical effects of supporting direct external change of system aspects might be.

Simulated annealing is not an inherent part of asynchronous teams, it is simply one technique for implementing contracting search. Contracting search is, however, a fundamental property of A-teams: it is the means by which asynchronous agents coordinate their efforts to find solutions.

6. References

[Donnett 87] Donnett J.G., *"Simulated annealing and code partitioning for distributed multimicroprocessors,"* Tech. Rep. 87-194, Dept. of Comp. and Info. Sci., Queen's Univ., Kingston, Canada, July 1987.

[Fenves 92] Fenves S., Flemming U., Hendrickson C., Maher M., Quadrel R., Terk M., Woodbury R., *"Computer Integrated Building Design"*, Prentice-Hall, 1992, to appear.

[Geman 84] Geman S. and Geman D., "Stochastic relaxation, Gibbs distribution and the Bayesian restoration of images," *IEEE Trans. on Patt. Anal. and Mach. Intell.*, vol. 6, pp. 721-736, Nov. 1984.

[Hastings 85] Hastings H.M., "Convergence of simulated annealing," *ACM SIGACT*, vol. 17, no. 2, pp. 52-63, Fall 1985.

[Huang 84] Huang M., Romeo F. and Sangiovanni-Vincentilli A., *"An efficient general cooling schedule for simulated annealing,"* Tech. Rep., Dept. of EECS, Univ. of Calif. Berkeley, Berkeley CA., 1984.

[Kirkpatrick 83] Kirkpatrick S., Gelatt C.D., Vecchi M.P., "Optimization by simulated annealing, *Science*, v. 220, pp. 671-680, May 13, 1983.

[Maher 87] Maher M.L., and Longinos P., "Development of an expert system shell for engineering design," *Int'l Journal of Appl. Eng. Education*, Pergamon Press, 1987.

[Metropolis 53] Metropolis N., Rosenbluth A.W., Rosenbluth M.N., Teller A.H., Teller E., "Equation of State Calculations by Fast Computing Machines," *Journal of Chemical Physics*, v. 21, pp. 1087-1092, 1953.

[Naher 85] Nahar S., Sahni S. and Shragowitz E., "Experiments with simulated annealing," *22nd Design Automation Conf.*, pp. 748-752, 1985.

[Quadrel 91] Quadrel R., *"Asynchronous Design Environments: Architecture and Behavior"*, PhD Dissertation, Department of Architecture, Carnegie Mellon University, Pittsburgh, PA., September 1991.

[Radford 88] Radford A.D., Gero J.S., *Design by Optimization in Architecture, Building and Construction*, Van Nostrand Reinhold, N.Y.C., N.Y., 1988.

[Romeo 84] Romeo F., Vincentelli A.S. and Sechen C., "Research on simulated annealing at Berkeley," *Proc. Int. Conf. on Computer Design*, pp. 652-657, Oct. 1984.

[Rutenbar 89] Rob. A Rutenbar, "Simulated Annealing Algorithms: An Overview," *IEEE Circuits and Devices Magazine*, pp. 19-26, January 1989.

[Slagle 89] Slagle J., Bose A., Busalacchi P., Park B. and Wee C., *"Enhanced simulated annealing for automatic reconfiguration of multiprocessors in space,"* Tech. Rep., Dept. of Comp. Sci., Univ. of Minnesota, Minneapolis MN., 1989.

[Talukdar 91] Talukdar S.N., deSouza P.S., Quadrel R., Ramesh V.C., "A-Teams: Multi-Agent Organizations for Distributed Iteration", in the *EPRI-NSF Workshop on the Application of Advanced Mathematics to Power Systems* , Redwood City, CA, Sept. 1991.

[White 84] White S.R., "Concepts of scale in simulated annealing," in *Proc. Int. Conf. on Comp. Design*, pp. 646-651, Oct. 1984.

[Woodbury 91] Woodbury R., "Searching for designs: paradigm and practice," *Building and Environment*, vol. 26, no. 1, pp. 61-73, 1991.

Appendix A: The simulated annealing algorithm

The simulated algorithm itself is straightforward. T_0 is the starting temperature, T_f is the freezing temperature. T is the annealing temperature, which is initially set to T_0, and is decremented by a factor of k3 during each iteration of the outer loop. A and R are the number of solutions that have been accepted and rejected, respectively. The inner loop repeats until either $A \geq k1$ or $R \geq k2$ (k1, k2 and k3 are predefined constants).

a. Calculate T_0 , the starting temperature, and T_f the freezing temperature.

b. Set $T \leftarrow T_0$.

c. Set $A \leftarrow 0$ and $R \leftarrow 0$.

d. Generate an initial solution S.

e. Calculate $C \leftarrow$ Cost(S).

f. While $T \geq T_f$ do

f1. While $(A < k1)$ and $(R < k2)$ do

- Generate a new solution S' by perturbing S.
- Calculate $\Delta C \leftarrow$ Cost(S') - Cost(Best S so far).
- If $(\Delta C \leq 0)$ or $(\text{random}(0,1) < e^{-\Delta C/T})$

then $S \leftarrow S'$, $A \leftarrow A+1$, $C \leftarrow C+\Delta C$

else $R \leftarrow R+1$

end While

f2. Set $A \leftarrow 0$, $R \leftarrow 0$, $T \leftarrow k3*T$

end While

The function random(0,1) returns a real number on the interval [0,1]. The conditional in statement **f1** allows a perturbation to be accepted with probability 1 if it generates a solution with a lower cost than the current one. If the new solution has a higher cost, it will be accepted with probability $e^{-\Delta C/T}$ (from [Donnett 87]).

The probability function

Nahar, Sahni and Shragowitz [Naher 85] have experimented with replacing the annealing's probability function $e^{-\Delta C/T}$ with other functions using C, ΔC and T. Instead of the

exponential function, they tried linear, quadratic and cubic functions of C. These were attempted both with a static temperature as well as step-wise decreasing temperatures. Their results show that the exponential function outperformed the other tested functions by a factor of two. Their explanation was that particles in real physical systems exhibit a statistical behavior that is closely modelled by the exponential function, implying that simulated annealing may be a "natural" approach to optimization [Donnett 87].

Cost function

The cost function is domain-dependent function that measures the value of one solution relative to another. The units of cost play an important role in the selection of a starting temperature, an example is White's standard deviation method(White 84). The cost difference in a move is also important. Examining the acceptance probability expression, $e^{-\Delta C/T}$, shows that to guarantee an acceptance probability of at least .85 at the starting temperature, we must use a temperature that has at least a 6:1 ratio to the "typical" difference in cost ($e^{-1/6} \doteq 0.846$). As the temperature:Δcost ratio decreases, so does the probability of acceptance. Beyond ratios of 1:6, there is an extremely low probability that poor solutions will be accepted ($e^{-6/1} \doteq 0.002$).

Starting temperature

The starting temperature should be relatively high, so that almost all perturbations will be accepted. This allows for a broad sweep of the search space, since it will accept solutions that have both low and high "costs."

Kirkpatrick suggests one approach for finding the starting temperature[Kirkpatrick 83]:

a. choose a starting temperature T at random

b. attempt a sample of perturbations at T, keeping track of the percentage of accepted (A) and rejected (R) solutions.

c. if A/(A+R)<0.8, then double the temperature and go back to b. Continue until system is "warm enough."

The problem is that this might give temperatures that are too warm, making the algorithm more time-consuming than it need be. White shows that a good heuristic for determining starting temperature can be derived from the standard deviation of the distribution of problem state densities vs. the costs of these states [White 84]. This requires performing enough sample moves to determine what the distribution is, however.

Annealing schedule

The annealing schedule determines both the conditions required for reducing the temperature as well as the factor by which it should be reduced. Formalization of the annealing algorithm has concentrated on annealing schedule, primarily because of its strong effect on the performance of the algorithm. Although White has proposed a general theory to decide the annealing schedule parameters [White 84], much of the variation of annealing schedules is based on empirical tests. Donnett suggests that schedules should be uniquely designed for the domain and the type of problem being addressed [Donnett 87].

Geman and Geman developed a "worst-case" schedule (impossibly slow) that guarantees convergence of the algorithm [Geman 84]. Hastings shows that annealing can converge faster than the worst case [Hastings 85]. Romeo et al. have proved the algorithm to be asymptotically optimal [Romeo 84]. The longer the algorithm runs, the better chances of finding an optimal solution.

The two questions of the annealing schedule are: What is the temperature decrement over time? What is the number of perturbation iterations at each temperature? Staying at the same temperature for a long period of time ensures the best solutions possible (since simulated annealing is asymptotically optimal, staying at each temperature for an amount of time t, where t approaches infinity, guarantees the global optimum will be found), but this is not feasible in practice.

Two methods are known which compute the number of perturbations at each temperature interval: the cost-based function method and the accept-reject count method [Slagle 89].

a. The cost-based function method suggests that the number of iterations N for any temperature T is given by:

$$N(T) = e(h_{max}(T)-h_{min}(T))/T \quad (1)$$

where $h_{max}(T)$ and $h_{min}(T)$ are the highest and lowest values of the cost function obtained thus far at this temperature [Huang 84]. In practice however, N(T) becomes too large as the temperature decreases, and requires the introduction of an (arbitrary) upper bound.

b. The accept-reject count method allows either a certain number of acceptances (k1) or a certain number of rejections (k2) at a certain temperature. These numbers are usually determined empirically (Donnett uses an accept/reject count in a 1:10 ratio [Donnett 87]). At each temperature, either k1 moves must be accepted or k2 moves must be rejected before the temperature is allowed to decrease.

Once the number of perturbation iterations is reached at a particular temperature interval, the temperature is decremented by some factor (k3). Choosing k3 to be less than 1 gives an exponentially decaying temperature. As the search nears completion, the temperature should decrease very slowly, giving the system the chance to find the optimal solution. Values for k3 that are less than 0.5 "cool" the system too quickly and results in less-than-best performance.

Donnett performed a series of annealing experiments in which k3 varied between 0.5 and 0.99 in increments of .01 His results show that values between 0.8 and 0.9 provide the best trade-off between execution time and achievement of optimal solutions [Donnett 87].

Terminating conditions

Annealing stops when the temperature reaches T_f. White [White 84] has found that the temperature at which no more improvement can be made is

$$T_f = (C1-C0) / \ln s \quad (2)$$

where C0 is the absolute minimum cost, C1 is the next largest cost, and s is the number of state perturbations that can take the search from C1 to C0. The freezing temperature calculated by this formula represents the most pessimistic terminating conditions: when using this value, even in the worst case, no anneal will stop too soon.

Another strategy for determining terminating conditions is the one used by a commercial simulated annealing package called TimberWolf™. This strategy halts the anneal when no new solutions have been accepted after four consecutive decreases in temperature. Slagle [Slagle 89] has shown that this strategy may stop the anneal too early, however Donnett [Donnett 87] has found that in many cases, his results "froze" before reaching the pessimistic freezing temperature. We used this heuristic in our implementation.

Multi-criteria simulated annealing

Our implementation of the simulated annealing algorithm follows from the pseudo-code given in above, but raises an important issue: How does one measure the "cost" of a design solution that is evaluated by multiple criteria? One technique is to "weigh" the various evaluations and sum them, but this introduces artificial and formally insupportable measures of importance to various features of the design. Instead of comparing designs on the basis of "cost," we use a *dominance relationship*.

Dominance relationships

The evaluation of a solution is given as a vector, where each vector element is a "score" calculated by a unique evaluator agent. These scores may be continuous or discrete values, but in either case there is a notion of a "better" or "worse" score (e.g., "the larger the score, the better the solution" or "the scores range from 1 to 5, with 1 being the best").

Rather than attempting to find a single, absolute value to represent a solution's quality, these evaluation vectors can be compared to measure a solution's quality relative to another's. There are three binary relations that exist between evaluation vectors (and hence, their corresponding solutions).

$S1 \Rightarrow S2$ S1 dominates S2. For all scores *i*, as *i* ranges from the first evaluation to the last, S1's *i*th evaluation score is better than or equal to S2's *i*th evaluation score.

$S1 \Leftarrow S2$ S1 is dominated by S2. For all scores *i*, as *i* ranges from the first evaluation to the last, S2's *i*th evaluation score is better than or equal to S1's *i*th evaluation score. Note: if $S1 \Rightarrow S2$ AND $S2 \Rightarrow S1$ then the evaluations of S1 and S2 are said to be equivalent.

$S1 \Leftrightarrow S2$ S1 contends with S2 iff S1 neither dominates nor is dominated by S2.

Using these relations, we can say that a solution S is non-dominated in a set of solutions X if there exists no solution $x \in X$ such that $x \Rightarrow S$. The set of all non-dominated solutions (the ND Set) comprise the "best" solutions produced for a given design problem (i.e, no solution can claim to be "better" than any ND solution). This set is commonly known as the Pareto set [Radford 88, ch.8].

The "traditional" simulated annealing algorithm keeps the most recent solution which is used to generate new solutions, and a single "best" solution against which these new solutions are compared. We, however, keep a population of "accepted" solutions from which further modifications may proceed. Also, instead of a single solution, we use the ND Set (a subset of this population) as the standard against which new solutions are compared. The following modified algorithm shows how we use simulated annealing for search in a space where solutions are evaluated by multiple criteria.

f. While $T \geq T_f$ do

f1. While $(A < k1)$ and $(R < k2)$ do

- Generate a new solution S' by
  modifying an *accepted* solution
- If $(S' \Rightarrow$ any X, for all $X \in$ ND Set)
  then accept S', replace all X with S', set $A \leftarrow A+1$
  else
  - If $(S' \Leftrightarrow X)$, for all $X \in$ ND Set)
    then accept S', add S' to ND Set, set $A \leftarrow A+1$
    else
    $D = \text{distance}(S', \text{ND Set})$
    - If $(\text{random}(0,1) < e^{-D/T})$
      then accept S', set $A \leftarrow A+1$
      else $R \leftarrow R+1$

end While

f2. Set $A \leftarrow 0$, $R \leftarrow 0$, $T \leftarrow k3*T$

end While

The idea here is nearly the same: A new solution S' is generated as a result of the recommendations of modifiers. If this new solution is as good (contends with) or is better (dominates) than any that already exist, then accept it immediately. Otherwise, compare it to the best solutions already obtained (using the distance function) and compute an acceptance probability based on this comparison and the current annealing temperature.

With the use of a an accepted set (of which the ND set is a subset), the issue of conflict resolution that typically arises in multiple agent systems is skirted. Conflicts can be thought of as being produced by design changes that make some evaluations better and others worse. In the above algorithm such designs can readily be produced. Designs that contain conflicts will either take a place in the ND set as designs that do not dominate other designs or they will tend to be evaluated poorly, i.e., they are likely to be dominated

by other solutions in the accepted set and thus will only enter the accepted set under the probabalistic criterion of the simulated annealing algorithm..

The distance function

The distance function provides a means to compare "how far off" a candidate solution is from the "best" solutions that are currently in the ND Set. Consider an N-dimensional Euclidean performance space, where each axis of the space corresponds to a single evaluation criterion. Each solution can be plotted as a point in this space, based on the values listed in its evaluation vector. Now consider an imaginary solution in which each element of its evaluation vector contains the best value found for that criterion after examining all existing solutions. This imaginary solution is called the target solution and its evaluation vector is the target vector.

The target solution is a reference point that can be used to determine "how far off" the candidate solution is from the ND Set. The distance function we used computes this value as the difference between the distance from the candidate to the target, and the distance from the ND Set to the target. This function reduces the estimated distance in situations in which there is a large angle between a target-candidate vector and a target-ND-set-element vector. It has the effect of making it more likely to accept a solution in parts of the performance space that have been sparsely visited. The target-to-ND-set-element calculation can become somewhat tricky: how does one compute a distance between a point and a set of points? Again, we don't want to resort to a scheme that selects some "average" point by introducing artificial weights.

In the current implementation, we simply choose a single solution from the ND Set that is closest to the candidate, and performs the distance calculations using this solution. A more accurate method would be to compute a Minkowski distance measuring the length of a projection from the target solution to the Pareto surface defined by the ND Set. That surface could be estimated by an N-dimensional hyper-plane that locally approximates the Pareto frontier.

Perturbation functions

Solutions are perturbed by the actions of modifier agents. Given a candidate solution that can be improved, modifiers can choose between two options: either modify the solution directly, or modify an input to a generator that will, in turn, produce a new solution on its own.

Although the first method seems most direct, it can lead to consistency problems. Once modified, there is no way of knowing whether the new child aspect is consistent with respect to its parent aspects. Agents could be constructed to verify these dependency relationships, but since flexibility is a high priority for asynchronous design environments, it would contradict the philosophy of A-teams to demand the existence of such agents.

The second option, modifying the inputs to generators, is called problem restructuring. The modifiers essentially create a new problem for the generators to solve. By adding, deleting or modifying constraints, modifiers can guide the activity of generators to produce better solutions; and at the same time avoid the consistency problem.

Annealing schedule

The annealing schedule to a large extent determines the performance of the simulated annealing algorithm. The schedule specifies the starting temperature and how quickly that temperature decreases. It also specifies (by using the accept-reject count method) how long the environment remains at one particular temperature level. The values that were used in the experiments described in Section 4 of this paper are:

starting temperature = 180° (estimated)
freezing temperature = 5° (estimated)
accept ceiling (k1) = 4
reject ceiling (k2) = 8
temperature decrease factor (k3) = 0.85

The starting temperature was selected based on the results of tests performed before the experiment began. By producing a number of solutions and computing their distances

from a target solution, we were able to estimate that "typical" distances ranged between 18 and 27 units. From this, we selected 180° as a the value for starting temperature to match the desired 6:1 temperature:distance ratio described in above. By similar reasoning we estimated the freezing temperature to be 5°.

The accept/reject ceilings were selected based on success with these values in previous trials of the simulated annealing algorithm. After our experiments, however, it would appear that these values may be too low - our suspicion is that the system may have performed better if it were able to remain at each temperature interval for a longer period of time.

The temperature decrease factor of 0.85 was chosen based on the empirical results of Donnett's work [Donnett 87].