Structural stigmergy: A speculative pattern language for metaheuristics

Ben Kovitz
Fluid Analogies Research Group
Indiana University
512 North Fess Avenue
Bloomington, Indiana 47408 USA
bkovitz@indiana.edu

Jerry Swan
Computing Science and Mathematics
University of Stirling
FK9 4LA Scotland UK
jerry.swan@cs.stir.ac.uk

ABSTRACT

To construct graphs whose quality results from complicated relationships that pervade the entire graph, especially relationships at multiple scales, follow a strategy of repeatedly making local patches to a single graph. Look for small, easily recognized flaws in local areas of the graph and fix them. Add tags to the graph to represent non-local relationships and higher-level structures as individual nodes. The tags then have easily recognized flaws that relate to non-local and higher-level concerns, enabling local patching to set off cascades of local fixes that address those concerns.

Categories and Subject Descriptors

I.2.8 [**Problem Solving, Control Methods, and Search**]: [Heuristic methods]; G.1.6 [**Optimization**]: [Stochastic programming]

General Terms

Algorithms, Design

Keywords

metaheuristics; stigmergy; design patterns

1. FIVE PATTERNS

1.1 Local patching

In search problems where each candidate solution is a graph in which the goodness of a solution results from complicated relationships that pervade the entire graph, **often it's easy for a person to identify local flaws and ways to fix them but hard to design a fitness function that provides a workable gradient to go from poor to good solutions.**

For example, an analog circuit is a complicated graph in which the nodes are electronic components and the edges are electrical connections between those components. Some simple flaws in a circuit design: an integrated circuit's V_{CC} pin isn't connected to the supply voltage; an electrolytic capacitor is connected in reverse polarity; a component isn't connected to anything at all. These flaws all have simple and obvious fixes. But the relationships between components to make a correct or acceptable circuit are subtle and complex: waveforms and timings result from feedback loops between components and must be chosen correctly to orchestrate the activity of other components, etc. Almost any random variation made to a circuit renders it completely inoperable. Consequently, it's hard to design a fitness function without large plateaus and bad local minima.

Therefore:

Instead of maintaining a population of graphs, **maintain a single graph. Make a "library of patches": a collection of types of small, local flaws together with simple ways to fix them. On each iteration of the search, choose one flaw randomly and fix it.**

Each flaw should be computationally cheap to recognize: ideally, by examining a single node and its immediate connections. No flaw can pertain to a single node alone. Every flaw should pertain to a node's relationship to its neighbors.

There should be more than one way to fix many of the flaws. For example, a completely unconnected component simply needs to connect to *something*. Make the system choose randomly among the fixes available in the circuit as it stands now.

Ideally, each fix consists of an operation on a single node or edge: creating, modifying, or deleting a node, or creating, modifying, or deleting an edge connecting two nodes.

Consequences:

1. Most changes improve the graph.

As the search runs, consecutive changes appear scattered about the graph, each bringing the graph closer to a good graph. But they don't necessarily result directly in a graph that, considered as a whole, is any better. For example, even after making a number of connections, a circuit design might still not work at all.

2. Fixing one flaw often creates a new flaw.

For example, hooking up an LED to a positive voltage and ground without also adding a pull-up resistor is likely to send too much current through the LED.

3. The detected flaws become progressively more specific to their context.

For example, if an electrolytic capacitor wasn't connected to anything at all, detecting that flaw might trigger the

GECCO'14, July 12–16, 2014, Vancouver, BC, Canada.

ACM 978-1-4503-2881-4/14/07 ...$15.00.
http://dx.doi.org/10.1145/2598394.2609845.

patch "hook it up to any component that can connect with a capacitor." Once connected to one component, its flaw is that it needs another connection, having something to do with the other component. Once connected to two components, the capacitor might be connected with the wrong polarity, triggering a patch to reverse its connections.

4. There's no tendency to converge on a particularly good graph, only a "reasonable" graph.

This may be OK for some searches, such as a search for a valid solution which can be defined entirely in terms of local relationships between nodes. For many searches, though, we want to find better and better graphs as defined by some set of criteria, such as operating characteristics of a circuit. See "Goal tags" (sec. 1.4) for how to address this.

1.2 Change creates salience

Often a sequence of several small changes in one place is needed to do much good. But the first change doesn't get followed up because the attention of the system is distributed equally everywhere.

For example, a voltage regulator needs a capacitor and resistor, a power source, and an output which matches the voltage needed by the integrated circuit(s) whose voltage it regulates. The regulator's output is itself a function of the capacitor and resistor that it's connected to. If only one of these fixes is done and then ignored, the other components will have flaws of their own, which might trigger incompatible fixes.

Therefore:

Associate a level of salience with every node. Give it a boost whenever that node or its edges are modified. And let it decay on every search step when it's not modified. When randomly choosing nodes to fix, give higher probability to the more-salient nodes.

When a node has no flaws, it won't get modified, providing opportunity for decay. A node also won't be modified while attention is on far-off nodes in the graph, even if that node has a flaw.

Consequences:

1. Flaw-fixing tends to happen in local cascades.

Fixing one flaw tends to create a new flaw in the same node or an adjacent node. That flaw triggers another fix, and so on.

2. Sometimes the system flounders, undoing a fix it just made for one node because of the flaw that it created for a neighboring node.

Floundering is not necessarily bad. The resulting instability creates a variety of opportunities for other nodes to make a connection with one of the floundering nodes. This fixes more of the node's potential flaws, resulting in greater stability. To counter floundering of a more pernicious sort, see "When you're stuck, break something" (sec. 1.5).

3. Distant parts of the graph can build up internally coherent but mutually incompatible subgraphs.

This is partly addressed in the next pattern, and partly in "When you're stuck, break something" (sec. 1.5).

1.3 Tags

Most relationships of interest are not local. Making a good decision regarding one node requires knowledge of how that node fits into a larger structure, and even how that larger structure fits into yet larger structures.

Therefore:

Attach tags to nodes, groups of nodes, and nodes with an important relationship between them.

Tags are "administrative" nodes, which represent non-local knowledge where it's needed. You must define a "library of tags" and situations they apply to. And you must extend the "library of patches" to include flaws involving tags and what's needed to fix them.

For example, a group of several electronic components might produce a certain waveform. This group should be tagged as "generating a waveform". It's a flaw for this tag to lack an appropriate power source, to lack a receiving component or module into which to feed the waveform, or for its frequency to be unsuitable for that required by the receiving component.

If certain kinds of nodes must be arranged in a certain sequence, tags can indicate where in the sequence they occur. For example, it's a flaw for an English sentence to end with "the", so, in a program that constructs sentences, the last word in a sentence might get a special tag.

In some problems, tags can themselves be part of the solution in addition to performing an administrative role (e.g. Copycat, sec. 3.1).

Consequences:

Relationships between aspects of the graph that exist on a larger scale than a single node drive the search.

A tag that binds several nodes into a group can have neighbors that are also tags, which bind other groups together. Now, from the standpoint of local patching, the high-level structures and their relationships are local, too. Tags that designate relationships of interest can have tags as neighbors, and consequently flaws in relation to those neighbors, and consequently set off new cascades of local patching.

1.4 Goal tags

Internal coherence alone is not enough to make a practical metaheuristic. There must be a way to drive the graph toward predefined objectives.

Therefore:

Represent objectives with "goal tags": tags that are flawed when they don't connect with nodes that implement the objective.

For example, a goal tag for a circuit to generate a sine wave would be flawed when it lacks connection to a tag that describes an output waveform. Upon connection with such a tag, there might still be a flaw because the waveform doesn't match the shape or frequency specified in the goal tag. This flaw sets off a cascade of changes to the waveform tag's other neighbors.

Consequences:

The goal tags provide "top-down" pressure to direct the search, balancing the opportunistic "bottom-up" pressures that come from looking for ways to connect the lowest-level nodes.

At the start of a search, the goal tags' flaws can't be satisfied, because the tags representing the high-level structures and relationships they need to connect to don't exist. To fix this flaw, the program makes a tag for the relevant kind of structure or relationship, just without the structure or

relationship existing. This is a flaw for the new tag, which creates pressures to build up the needed kinds of structures.

1.5 When you're stuck, break something

Sometimes, two large parts of the graph are each internally coherent but they conflict with each other, and there's no way to tell which one is "wrong".

In this situation, most of the nodes have few flaws left to fix, except the goal tags are not satisfied or only poorly satisfied. Each local patch seems to do as much damage as good. Small changes to either part of the graph make that part locally worse. Neither part of the graph offers a path of locally beneficial changes to produce a global improvement.

Therefore:

Monitor the rate of local improvement throughout the graph. When it stops, arbitrarily unlink or destroy a large, connected part of the graph.

Since there's no way to tell which part should be destroyed, the best you can do is arbitrarily destroy one of them, so something new can grow in its place, hopefully influenced by and more harmonious with the remaining part.

On the other hand, if you have enough domain knowledge to program an educated guess about which part should be destroyed, you can implement "competition". When a node tries to fix its own flaw by linking to a node that's already linked, it must dislodge the competitor. The system can give varying weights to the first node's fix, the creation of a flaw for the competing node, and the effect on the node connected to. The winner is determined by these weights; the loser is disconnected and will have to find some other way of getting the connection it needs. Or if the loser is a group, the group tag can be removed, releasing the contained nodes to build a fresh structure. The weights can reflect how long a connection has been in existence, how long each node has been flawed, and how serious the flaws are.

Consequences:

The search cycles between slow construction and fast destruction.

With competition, subgraphs tend to prevail if they can contribute to the global goals, and tend to be destroyed if they can't.

There is, however, no way to tell in advance if breaking up a large part of the graph will lead to something better or not. It's possible for the program to find the best graph it will ever find, break it up because it can't improve it, and never find it again.

2. CONSEQUENCES FOR THE DESIGNER

1. The problem becomes more tractable for the metaheuristic designer.

Local flaws are very easy to understand and think of how to fix. As you watch the search in progress, you see simple flaws that are not being addressed, and you think of new additions to the library of patches and the library of tags. In effect, you patch up flaws in the metaheuristic as you find them, not so different from what the metaheuristic does. Criticizing, nit-picking, and patching is much easier than coming up with an elegant theory.

You never have to invent a fitness function. You just define various kinds of connections that you want to see in a solution graph.

2. You can add all the domain knowledge you like.

For example, a circuit-design metaheuristic can exploit as much electronics knowledge and circuit-design know-how as you can find. You keep adding it a little bit at a time, by translating it into the language of tags and flaws.

3. Because of that, the metaheuristic tends to acquire both human-like common sense and human-like blind spots.

Genetic algorithms sometimes discover bizarre circuits that no human engineer ever would—like circuits where one part is not connected but that don't work if the unconnected part is removed [11]. This is both good and bad, but in practice it's usually bad. The good is that such searches don't have the blind spots that sometimes prevent humans from seeing excellent solutions. The bad is that bizarre solutions often exploit dependencies that make them impractical. For example, the above-cited circuit worked by exploiting thermal properties resulting from the manufacturing quirks of a single FPGA chip. When tried in another FPGA of the exact same model, the circuit design didn't work.

4. You must manually adjust global parameters.

Boosting salience, decaying salience, weighting the importance of flaws, determining when local improvement has petered out—all these involve choosing constants with far-reaching effects on the performance of the program. If these global parameters aren't just right in relation to each other, the system can spend all of its time stuck on a bad solution, abandon excellent solutions, break up nascent solutions before they've had a chance, etc. This sounds like the kind of problem that a metaheuristic could help solve. To date, though, programs that do structural stigmergy with local patching (sec. 3) have gotten their global parameters set by somewhat laborious manual tuning.

3. EXAMPLES

Because these are speculative patterns, there are few examples of metaheuristics that use them. These patterns are found mostly in biology (sec. 4) and in blackboard systems [2]. Blackboard systems model collaboration of different experts on the solution to a problem. All can see and edit the current version of the solution. We briefly describe Copycat, a blackboard-like system that demonstrates local patching and tagging especially well.

3.1 Copycat

The Copycat program [8] makes analogies between strings of letters. Given an example of changing one letterstring to another, such as `abc` → `abd`, Copycat will attempt to "do the same thing" to another given string. Attempting to make the analogous transformation to `iijjkk`, it might come up with `iijjll`, `iijjkl`, `iijjdd`, `ijl`, or other choices, depending on how it mapped the example to the given string and what rule it came up with to explain the example.

Flexible analogy-making faces problems fundamental to all metaheuristics: the number of possible choices is vast, only a fairly small number of them are reasonable, it's hard to tell whether a bad candidate solution is near or far from a good one, and one must often settle for a solution that is less than ideal. In addition, there is no way to define a simple fitness function for all possible letterstring analogies. The set of possible relationships is just too vast and irregular.

Copycat makes extensive use of tags, in particular tags that define groups and tags that define sequences in terms of successor or predecessor relationships. Given the example

pqqp → qppq and the start string gghijj, Copycat could tag gg and jj as groups. This would influence the choice of mapping from pqqp, perhaps favoring mapping the first p to the gg and the second p to the jj, perhaps leading to a result of hggjji. Or Copycat might tag the successor relationships gg→h→i→jj, leading it to tag the initial string with successor/predecessor relationships: p→q←q←p. It might then convert the input string to an analogously symmetric "hill": gghijjjjihgg.

All the parts of the solution grow together, each tag setting off searches for ways to make an analogous tag elsewhere or ways to define a mapping rule to make use of the tags. Tentatively considered mapping rules set off searches for appropriate ways to tag the letters. Nodes become "unhappy" due to inappropriate positioning in the solution graph, triggering searches for ways to repair the problem. Competitions (as in sec. 1.5) determine when one node can dislodge another. A global "temperature" tracks overall satisfaction with the current solution, and affects the probability of a "breaker" randomly destroying part of the solution.

A spreading-activation network called the "slipnet" directs the search top-down, complementing the bottom-up search of nodes looking for appropriate links. When a node in the slipnet becomes active, the corresponding type of node in the "workspace" (the tentative solution now being worked on) is deemed most relevant to the search right now, giving actions ("codelets") related to nodes of that type higher urgency. The slipnet also models something not covered above: the ability of concepts to "slip", or adjust their meaning to fit what's currently feasible to achieve in the workspace.

4. BIOLOGICAL INSPIRATION

4.1 Stigmergy

The biological phenomenon of stigmergy has been a fertile source of inspiration for metaheuristics [3]. Stigmergy is the stimulation to act resulting from effects in the environment of actions already completed. For example, when termites are building a nest, the parts of the nest already built stimulate the termites to perform the next steps toward completing the nest. Each insect decides what to do next only by sensing stimuli that are present and nearby. Pheromones left by ants while searching for food are a natural tag: a sign left in the environment rather than simply a side-effect of action.

Ant-colony optimization [4] was, of course, inspired by ants—as was Copycat [5]. Stigmergy in ant-colony optimization is limited to attaching tags ("pheromones") to edges of a graph in order to steer the search toward favorable variations. We define the term *structural stigmergy* here to mean specifically the building of hierarchical structures by local patching. This requires both building upon or correcting partial structures and tagging partial structures to make non-local information local.

John Holland has discovered a wide variety of tags in complex adaptive systems, especially tags that define or distinguish groups ("aggregates"). A particularly relevant example is the use of a flag to rally an army or political group [6, p. 13]. The group members' attachment to a symbol enables them to be led as if they were a single entity. Yet each member makes locally appropriate decisions, helping the group goal according to his or her own skills and opportunities.

4.2 Stigmergy in human life

Stigmergy occurs in human life at least as much as it occurs in social insects [10]. In addition to natural side-effects of action, such as trails of worn vegetation left by walking and the spread of information in a marketplace through individual negotiations, humans deliberately leave signs in their environment that represent information in terms of conventional symbols. For example, road signs direct traffic, advertisements connect buyers with sellers, and of course all writing is signs deposited in the environment to carry information. We should not be surprised that when people notice something wrong in other people's writing, they often feel an immediate urge to correct it [9].

Wikipedia is made entirely by stigmergy. Each editor edits without consulting others, stimulating other editors to make corrections or improvements [7]. Wikipedia articles have "tags" explicitly noting the presence of problems: non-neutral point of view, lack of cited sources, etc. These tags stimulate other editors to correct the problems noted.

Christopher Alexander's concept of a "pattern language" is itself a form of structural stigmergy. Each pattern is a way of solving a recurring problem: a conflict between forces (drives to action) that arises in a situation [1, ch. 14]. Building is a process of repair [1, ch. 24], making local changes to fix local problems.

The enormous success of structural stigmergy in nature suggests that it's both workable and effective for many hard problems. It demands little at each decision-point, and it produces structures that cohere at all levels and fit external pressures such as an environment or a set of objectives.

5. REFERENCES

[1] Alexander, C. *The Timeless Way of Building.* Oxford University Press, 1979.

[2] Craig, I. *Blackboard systems.* Intellect Books, 1995.

[3] Crina, G., and Ajith, A. Stigmergic optimization: Inspiration, technologies and perspectives. In *Stigmergic optimization.* Springer, 2006, pp. 1–24.

[4] Dorigo, M., and Stützle, T. *Ant Colony Optimization.* Bradford, 2004.

[5] Hofstadter, D. R., Mitchell, M., et al. The Copycat project: A model of mental fluidity and analogy-making. *Advances in connectionist and neural computation theory 2*, 31-112 (1994), 29–30.

[6] Holland, J. H. *Hidden order: How adaptation builds complexity.* Basic Books, 1995.

[7] Lih, A. Long live Wikipedia? In *A Companion to New Media Dynamics*, J. Hartley, J. Burgess, and A. Bruns, Eds. Wiley, 2013, pp. 185–190.

[8] Mitchell, M. *Analogy-Making As Perception: A Computer Model.* MIT Press, 2002.

[9] Munroe, R. Duty calls. https://xkcd.com/386/. Accessed: 4-Apr-2014.

[10] Parunak, H. V. D. A survey of environments and mechanisms for human-human stigmergy. In *Environments for Multi-Agent Systems II.* Springer, 2006, pp. 163–186.

[11] Thompson, A., Layzell, P., and Zebulum, R. Explorations in design space: Unconventional electronics design through artificial evolution. *Evolutionary Computation, IEEE Transactions on 3*, 3 (1999), 167–196.