Iowa State University
Digital Repository

Graduate Theses and Dissertations | Iowa State University Capstones, Theses and Dissertations

2009

Asynchronous stigmergic sorting of binary matrix patterns: applications of classical distributed computing ideas

Neeraj S. Khanolkar
Iowa State University

Follow this and additional works at: https://lib.dr.iastate.edu/etd

Part of the Computer Sciences Commons

Recommended Citation
Khanolkar, Neeraj S., "Asynchronous stigmergic sorting of binary matrix patterns: applications of classical distributed computing ideas" (2009). *Graduate Theses and Dissertations*. 11111.
https://lib.dr.iastate.edu/etd/11111

This Thesis is brought to you for free and open access by the Iowa State University Capstones, Theses and Dissertations at Iowa State University Digital Repository. It has been accepted for inclusion in Graduate Theses and Dissertations by an authorized administrator of Iowa State University Digital Repository. For more information, please contact digirep@iastate.edu.

Asynchronous stigmergic sorting of binary matrix patterns: applications of classical distributed computing ideas

by

Neeraj Suhas Khanolkar

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Science

Program of Study Committee:
Soma Chaudhuri, Major Professor
Giora Slutzki
Pete Boysen

Iowa State University

Ames, Iowa

2009

Copyright © Neeraj Suhas Khanolkar, 2009. All rights reserved.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

First off, I would like to thank my advisor Dr Soma Chaudhuri for providing me with a thorough introduction to the theory of distributed systems, and then allowing me to explore the exciting areas of biological complexity research using this theoretical foundation. Her gentle but firm pressure helped keep my work on track in unfamiliar territory, and rescued me several times when I was stuck during my research and writing.

I would also like to thank the other members of my committee: Dr Giora Slutzki and Dr Pete Boysen.

Dr Slutzki gave me detailed feedback on earlier drafts of this thesis, and was extremely encouraging in his comments. His motivating words reassured me that this exploration was indeed a worthwhile endeavor. Access to his mathematical knowledge and ideas provided a great boost to my work.

Dr Boysen showed great patience and understanding while I was making up my mind about which research direction to follow. His gentle inquiries into my progress also helped keep me on track.

I would also like to express my gratitude to Dr Gary Leavens for his many years of sagacious insights into programming languages, software engineering and the philosophy of computer science in general. His suggestions regarding technical writing and LaTeX have proven extremely useful to me.

I owe many thanks to Dr Raji Joseph for her feedback, encouragement and support throughout my work.

Finally, I am truly indebted to my parents and my brother (with his wonderful family) for their love, patience, support and understanding.

ABSTRACT

Multi-agent stigmergy forms the basis of explanatory theories for various self-organized biological phenomena, and also serves as an implementation strategy for several important artificial applications. While a number of sophisticated techniques have been used in the modeling and analysis of stigmergic processes, none of them, yet, seem to be drawn from the toolbox of classical distributed computing. Our goals are to investigate and lay the groundwork for the use of classical distributed computing ideas in reasoning about stigmergic computation.

Specifically, we investigate case studies that are drawn from the domain of binary matrix pattern sorting. Here, a 'swarm' of memory-less and non-communicating agents follow a set of local stigmergic rules to asynchronously sort the binary states of cells in a 2-D grid so as to satisfy some global pattern specification. This domain is attractive as a test-bed because it serves as an abstraction for instances of biological pattern sorting and also because of its stigmergic expressiveness.

We demonstrate the application of the following four distributed computing concepts: (1) execution serializability, (2) local checking, (3) variant functions, and (4) indistinguishability (the last as an impossibility proof technique) in the modeling and analysis of our case studies.

Based on our preliminary experience with this particular domain, it seems to us that classical distributed computing techniques could be applied further in reasoning about stigmergic systems, perhaps leading to the formulation of a generalized stigmergic computational paradigm based on the principles of distributed computing.

CHAPTER 1. Introduction

The term *Stigmergy* was coined by Pierre-Paul Grassé, to refer to the process of cooperative nest construction by termites [6]. It proposed that the nest under construction itself acts as a coordination mechanism for the worker termites, rather than any direct communication between them. Workers get stimulated by particular local configurations of the construction, and in response transform the current configuration into a new configuration, which in turn may trigger another transformation response from the same or some other worker in the colony. This process repeats until there are no more stimulating configurations, at which point the construction stops.

Today, stigmergy or stigmergic computation denotes any abstract cooperative task performed by a multi-agent system in which (a) the past modifications of the environment by the agents guide their current actions, (b) the agents individually follow local rules that taken together have the desired global effect, and (c) the agents have a limited local view of the environment [1]. The agents are usually very loosely coupled, with almost no direct communication between them and posses very limited individual memory. While these limitations make global coordination more difficult, it makes such agent 'swarms' quite robust and scalable. Any failed agent can be easily replaced by a new working agent that has identical local rules as part of its 'DNA'. Since the original agent had no memory nor has sent any direct agent-to-agent messages, its failing does not lead to any significant loss of acquired information or disruption of the task in progress. Similarly, because agents are unaware of each other, it is easy to introduce new agents into an ongoing process to possibly speed up the task completion. To sum up, in a stigmergic computation, the environment, i.e. the domain of the task under progress, and the modifications made to it—structural and chemical (for example by insect pheromones)—act as the sole mechanism for mediating the actions of all the agents involved.

1.1 Our goals

Stigmergy has been used as an explanatory tool in the analysis of various biological phenomena involving self-organization [3]. It has also seen applications in artificial domains like collective robotics [15], and optimization problems [9]. The techniques used in modeling and reasoning about stigmergic processes include genetic algorithms [2], random walk models [14], and various statistical and stochastic methods [3]. However, to the best of our knowledge there has been no attempt, yet, to apply ideas and tools from classical distributed computing theory in the modeling and analysis of stigmergic problems. Our aim therefore, is to investigate the use of these classical ideas—in suitably adapted form—for the framing of, and subsequent reasoning about, stigmergic computational problems.

1.1.1 Motivation

Over the years distributed computing has matured into a field with a large arsenal of tools and ideas that could be applied to multi-agent systems. We are motivated by the intuition that the application of some of these ideas to stigmergic problems may reveal aspects that complement the insights yielded by more sophisticated techniques like stochastic analysis. This intuition is based on the observation that an autonomous multi-agent stigmergic system with its decentralized control has a lot in common with an asynchronous distributed computing system with limited shared memory.

1.2 Our contributions

As an initial step in validating our intuition, this paper presents several examples where we apply classical distributed computing concepts to reason about stigmergic problems. The examples are all drawn from a specific domain of stigmergic applications called “pattern sorting”. Before explaining the details of pattern sorting and the reasons for its selection as the domain of investigations, we would like to summarize our contributions. In this paper, we

- make explicit the assumptions about the individual and collective behavior of agents which (among other things) ensure the *serializability* [12] of each stigmergic execution (Sec. 2.4.3).
- adapt the notion of *local checking* [16] from self-stabilization (Sections 2.3.1 and 2.5.1).
- design and use *variant functions* [4] (again from self-stabilization), to reason about the correctness of a stigmergic solution (Sections 3.1.1.1 and 3.2.1.1).
- apply the general notion of *indistinguishability due to limitations imposed by local knowledge* [10], in order to prove impossibility results (Sections 3.2 and 3.3).

1.3 Informal overview of pattern sorting

It might be useful, for explanatory purposes, to think of our model as a kind of a puzzle that consists of light bulbs arranged in a rectangular grid on a vertical wall, with each bulb's ON/OFF switch located next to the bulb itself. The objective of the puzzle is to transform any given configuration of ON and OFF bulbs in the grid, into a configuration in which the spatial arrangement of the ON and OFF bulbs satisfies some pattern specification. There are additional restrictions which we will give shortly, but first we start with a couple of examples of pattern specifications (these examples will also be covered in detail in a later chapter).

Our first pattern specification is called the *corner grouped* pattern—CG for short. Informally, CG specifies that ON bulbs in any row must be to the left of any OFF bulbs in that row, and ON bulbs in a column must be above any OFF bulbs in the same column (See Fig. 3.1 in Sec. 3.1). Another example is called the *two layer* pattern—TL, which specifies that ON bulbs and OFF bulbs should form two horizontal layers in the grid, with the ON layer on top. A possibly mixed row of ON and OFF bulbs may separate the two layers, but all the ON bulbs in this mixed row must be to the left of all the OFF bulbs in the row (see Fig. 3.8 in Sec. 3.2).

Now we give the additional restrictions for this puzzle system. The puzzle is to be solved by a team of workers each of whom can move around non-deterministically over the grid, and who are subject to the following constraints:

1. Each worker can see and modify the ON/OFF states of a limited number of bulbs at a time—bulbs that are within a local neighborhood of some predefined size. Thus a worker can never view the entire global state of the grid at once.

2. Workers do not have access to the row-column coordinates of any bulb; however, they can detect when a bulb is at the edge of the grid and when it is in the interior. Thus a worker can tell if a particular bulb is, or is not, at the top, right, bottom or left edges (or the respective corners) of the grid.

3. Workers cannot communicate with each other directly, nor can they keep any record of their past actions; they carry out their respective local modifications in a completely asynchronous manner w.r.t. the actions of other workers.

4. Two concurrent local modifications (by two different workers) cannot overlap in their local neighborhoods i.e. their local sets of bulbs must be mutually disjoint.

5. Each local modification must preserve the total number of ON (and OFF bulbs) in the grid.

The last constraint is why we refer to our problem as "pattern sorting" and not "pattern formation". A worker can only swap around the states of bulbs within the current local neighborhood, and cannot arbitrarily turn them ON or OFF. We now briefly detour into an explanation of our results in terms of this puzzle analogy, before returning back to connect it to our actual model.

We first semi-formally describe the behavior of workers, (both as individuals and as part of a team) so as to clearly state some simple assumptions that confer several desirable properties on our system. We then formally characterize the properties of patterns that can be sorted in such a puzzle system by adapting the idea of local checking [16] from self-stabilizing systems to our stigmergic context. Next, we prove that pattern CG can be sorted in such a system; this proof uses a cost measure that is based on the concept of variant functions [4] (also from self-stabilization literature). Then we give an impossibility proof using the general notion of indistinguishability to show that TL cannot be sorted in such a system (we also apply this technique to another pattern instance not covered here in the introduction).

Returning back to our puzzle board, imagine that each bulb also has a paper and pencil next to it, so that workers can now annotate each bulb with some label that serves as a sign to guide the future actions of other workers (or even itself). With this additional power it is now possible to sort pattern TL, and we see one such solution using 4 labels later in this paper. Next, we justify the choice of this domain.

Why is our pattern sorting domain a good test-bed for stigmergic investigations? There are two main reasons for this. First, there are several biological examples that can be described as self-organized pattern sorting such as, for example, embryonic cell sorting [13], and brood sorting in ants [5]. Our domain serves as a compact abstraction of such examples and future investigations could perhaps lead to insights into these phenomena. Secondly, two forms of stigmergic interactions have been observed in nature: one is mediated by a functional and structural modification of environment, and the other by the addition of non-functional information in the environment. Our model, with its clear distinction between visible states and auxiliary states (non-functional w.r.t. patterns), allows both kinds of stigmergic interactions to be effectively expressed.

1.3.1 Connection to the formal model

To make our formal model easier to understand we now informally show the correspondence between the various parts of this puzzle analogy and the components of our actual model. Instead of a grid of light bulbs we have a 2-dimensional matrix of triples which we call the *tri-state matrix*. Each entry, or cell, of this tri-state matrix is a triple that consists of (1) a "visible" binary state corresponding to the ON/OFF state of the bulbs, (2) a "positional" state that indicates whether the cell is located at one or more edges (outer rows and columns) of the matrix or is in the interior, and (3) an "auxiliary" state which models the paper and pencil next to each bulb.

The team of workers is now a collective of identical memoryless agents that we call *stigbots*—short for stigmergic robots. Each stigbot is assumed to possess (a) sensors with which it can sense all three kinds of states for each cell in some local sub-matrix (analog of local neighborhoods of bulbs), and (b) actuators with which it can change the visible and auxiliary states of the cells in that same sub-matrix. We model the link between sensors and actuators abstractly as a set of rules, where each rule is a stimulus-response pair. The sensors detect when the states of cells in a sub-matrix matches a stimulus, and this activates the appropriate actuators causing them to change the states of the cells (in the same sub-matrix) according to the specification of the paired response for that particular stimulus. For example, stimulus-response rulesets for sorting CG and TL are given in Fig. 3.2 (Sec. 3.1) and Fig. 3.10 (Sec. 3.2) respectively.

1.4 Related work

The work closest to ours is the in-depth work by Yamins and Nagpal [19] dealing with pattern formation (but not patten sorting) in 1-dimensional cellular automata-like models. While their agents are the processors forming the pattern itself, and they do not seem to explicitly focus on stigmergic computations, nevertheless our notion of local checkability can be considered as the stigmergic analog of the local checkability as defined in their model. They carry out a much deeper analysis of local checkability, and prove that it is necessary and sufficient for local solvability in their model. Their main goal is an automatic global-to-local compiler for automatic ruleset generation, and they do not seem to use any classical distributed computing tools in their analysis.

Holland and Melhuish [7] investigate the stigmergic clustering and sorting of frisbees by physical robots using a very simple rule set. Inspired by biological examples, their aim is is to investigate the role that real-world physics plays in such self-organized activities.

Karsai and Pénzes [8] show how a simple set of building rules based on local information can be used to explain the construction of nests comprising of hexagonal cells (see Fig. 1.1) by paper wasps. Their building algorithms—using rules based on information about the age of the cells—are able to generate all forms of paper wasp nests found in nature.

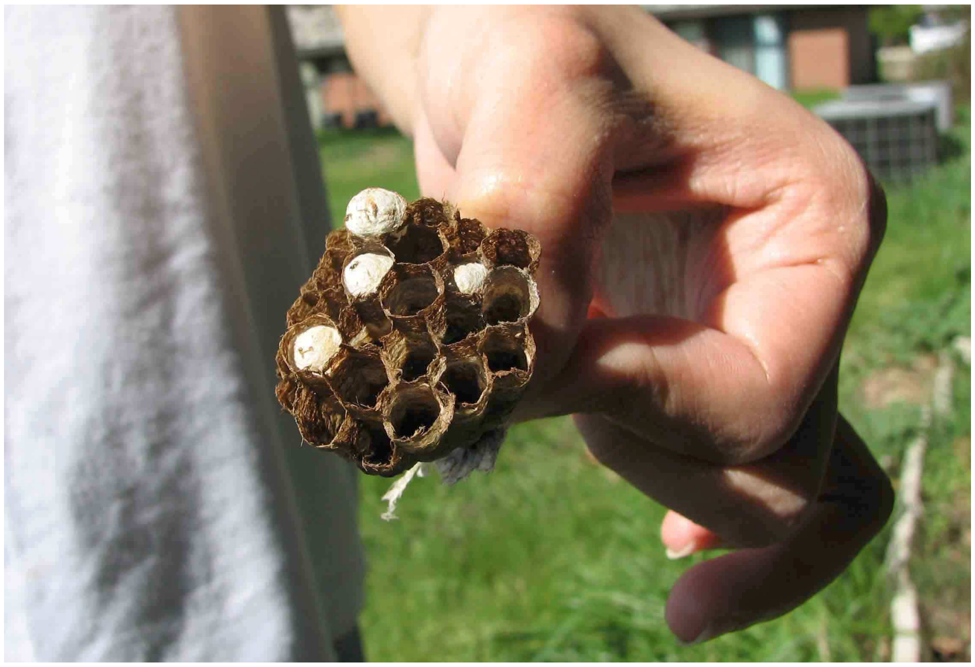


Figure 1.1 A paper wasp nest with (roughly) hexagonal cells.

Next, we look at a couple of works related to distributed construction and reconfiguration by robots using (various degrees of) stigmergy.

Werfel [18] describes solutions to the inverse problem of collective construction, where given a goal configuration, the robots autonomously decide where to attach each building block. Here the goal shape is specified by a coordinate-based map of all sites where a block should be attached. Each robot has access to this global shape map, and uses the local geometric features of the structure under construction as a reference (their basic version of stigmergy) to determine its own location relative to the shape map—in order to decide where to attach the block that it is carrying. Werfel then describes an extended form of stigmergy using building blocks that are more sophisticated in terms of the data storage and communication abilities. Using their own inter-block communications and the global shape map, these sophisticated blocks can now indicate to a passing robot as to where a block attachment is needed. The active entities here are thus quite powerful in terms of their computational and communication power, and access to global knowledge.

Walter et al. [17] describe distributed reconfiguration algorithms for metamorphic robots, where the goal is to have a chain of robots change its orientation in a plane. Here the robots themselves play the role of building blocks, and each robot can sense the presence or absence of its neighbors and can keep track of its own location w.r.t. some global coordinate system. Their robots move in lock step—i.e. are synchronized—and each has a fixed algorithm with a slight hint of stigmergy in that a robot's next step is based on the "stimulus" about the state of its neighboring cells.

CHAPTER 2. Our model

We now formalize our model which consists of configurations, patterns and stigbots. Since 2-dimensional matrices are at the core of our model, we start by defining some useful data structures and notation for general matrices that will be used later in the context of our tri-state matrices.

2.1 Rectangular domains

A *cell* of a $m \times n$ matrix, is any pair (i, j) where $1 \leq i \leq m$ and $1 \leq j \leq n$. We now formalize the grouping of cells that are bounded by a rectangle of certain width and height, within a matrix. If for some $m \times n$ matrix, we have positive integers i, j, p, q such that $1 \leq p \leq m, 1 \leq q \leq n, 1 \leq i \leq m - p + 1$, and $1 \leq j \leq n - q + 1$, then the *rectangular domain* $\mathcal{D}^{i,j}(p, q)$ denotes the set of all cells within the p-row and q-column rectangle that has cell (i, j) as its top-left corner, i.e. $\mathcal{D}^{i,j}(p, q) = \{(i', j') : i \leq i' \leq i + p - 1 \ \wedge \ j \leq j' \leq j + q - 1\}$.

The set of all $p \times q$ rectangular domains in a $m \times n$ matrix, is denoted by $\mathcal{D}^*_{m,n}(p, q)$ and is the set of all rectangular domains $\mathcal{D}^{i,j}(p, q)$ where $1 \leq i \leq (m - p + 1)$ and $1 \leq j \leq (n - q + 1)$.

Finally, the set of all rectangular domains (in a $m \times n$ matrix) with dimensions *at most* $p \times q$, i.e. at most p rows, and at most q columns, is denoted by $\mathcal{D}^*_{m,n}(\leq p, \leq q)$ and is equal to $\displaystyle\bigcup_{x=1}^{p} \left(\bigcup_{y=1}^{q} \mathcal{D}^*_{m,n}(x, y) \right)$.

2.1.1 Sub-matrices corresponding to domains

The value of a rectangular domain d in some matrix C is just the sub-matrix of C corresponding to the rectangle of cells in d, and is denoted $C(d)$. Formally, for a $m \times n$ matrix C, and a rectangular domain $\mathcal{D}^{i,j}(p,q)$ (that is valid for C), $C(\mathcal{D}^{i,j}(p,q))$ is a $p \times q$ matrix, say V, such that $\forall\ 1 \leq x \leq p,\ 1 \leq y \leq q : V[x,y] = C[i+x-1, j+y-1]$.

2.2 Configurations (tri-state matrices)

As described in the introduction, a *configuration* in our model is a 2-D matrix of triples, called a *tri-state matrix*, such that each entry is a triple in the set $\{0,1\} \times \mathscr{P}(\mathcal{E}) \times \mathcal{L}$, where $\mathcal{E}$ is a set of edge indicators, and $\mathcal{L}$ is a finite set of labels (more on these sets below).

For a $m \times n$ configuration C, we use C_V, C_P, and C_A to denote respectively the $m \times n$ matrices of the first, second and third components of C, i.e. C_V, C_P, and C_A are respectively the visible state matrix, the positional state matrix, and the auxiliary state matrix of C, where $\forall\ 1 \leq i \leq m, 1 \leq j \leq n$:

$C_V[i,j] \in \{\mathit{1}, \mathit{0}\}$. The number of *1* values in C_V is denoted by $\#_1(C_V)$.

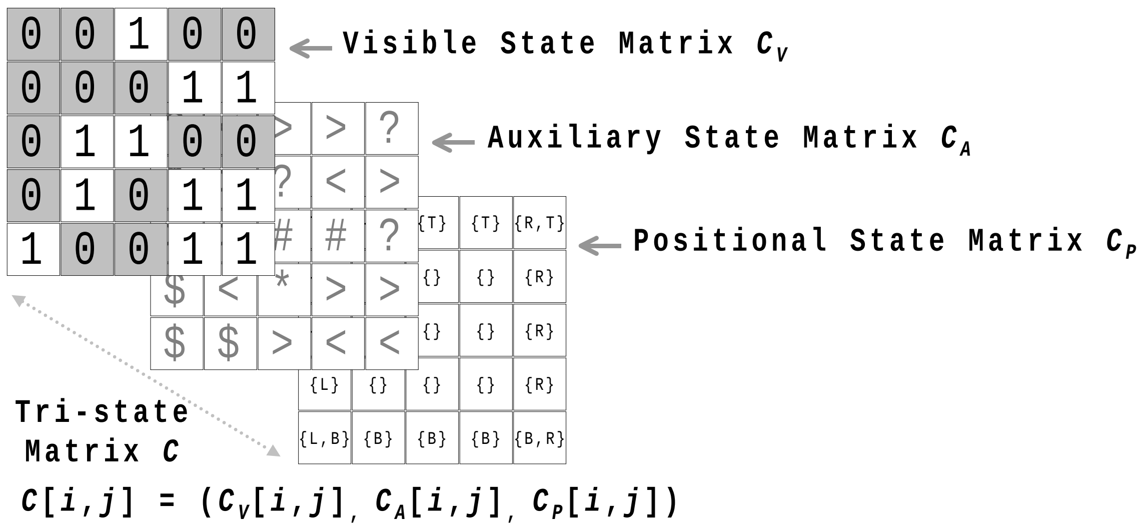
$C_P[i,j] \in \mathscr{P}(\mathcal{E})$, where $\mathcal{E} = \{$'*T*','*B*','*L*','*R*'$\}$. '*T*','*B*','*L*' and '*R*' represent positions relative to the boundaries of the matrix, i.e. respectively top edge, bottom edge, left edge and right edge. Thus '*T*' $\in C_P[i,j] \iff (i=1)$, '*B*' $\in C_P[i,j] \iff (i=m)$, '*L*' $\in C_P[i,j] \iff (j=1)$, and '*R*' $\in C_P[i,j] \iff (j=n)$. Clearly, $C_P[i,j] = \emptyset$ for all (i,j) locations in the 'interior' of the matrix.

$C_A[i,j] \in \mathcal{L}$ where $\mathcal{L}$ is a finite set of labels such that ' ' $\in \mathcal{L}$. The ' ' value is a default or nil label. If $C_A[i,j] =$ ' ' for each $1 \leq i \leq m$, and $1 \leq j \leq n$, then C_A is a *default auxiliary state matrix.*

Figure 2.1 shows a configuration as 'sliced' into its three component matrices, and Fig. 2.2 shows a 2 $\times$ 2 sub-matrix of this configuration.

2.3 Patterns

We define a pattern as a unary predicate on visible state matrices. For a pattern P and a configuration C, $P(C_V)$ is true if the arrangement of binary values in the visible state matrix C_V is consistent with the specification of P. We say C *is a P-configuration* if $P(C_V)$ holds.



Visible State Matrix C_V
Auxiliary State Matrix C_A
Positional State Matrix C_P
Tri-state Matrix C
$C[i,j] = (C_V[i,j], C_A[i,j], C_P[i,j])$

Figure 2.1 Slicing a configuration into its three component matrices.

The various patterns we consider in this work will be specifications of spatial constraints on the arrangement of *1* and *0* values in the visible state matrix of a configuration.

We call a pattern P *universal* if for any given row-column dimensions there is always a configuration satisfying the pattern, irrespective of the total number of *1* cells, i.e. $\forall\, m, n \in \mathbb{Z}^+, \forall\, 1 \leq t \leq m{\cdot}n : \exists\, m{\times}n$ configuration $C : (\#_1(C) = t) \wedge P(C_V)$. All the patterns considered in this work are universal; henceforth when we write “patterns” we mean “universal patterns”.

2.3.1 Locally checkable patterns

We now adapt the idea of local checking from self-stabilizing network protocols [16], into our stigmergic context. A network protocol is locally checkable if whenever the protocol is in a bad state, some link subsystem (i.e. a pair of neighboring processors and the channels between them), is also in a bad state. To apply this idea in the pattern sorting domain, we ask the question: for a pattern P, given any configuration C, is it possible to determine that C does not satisfy P by looking at the cells in just some small localized area of C? In other words, is there a sub-matrix of at most some fixed dimension, in any configuration, that can indicate if P does not hold in that configuration? If so, then the pattern is locally checkable. We make

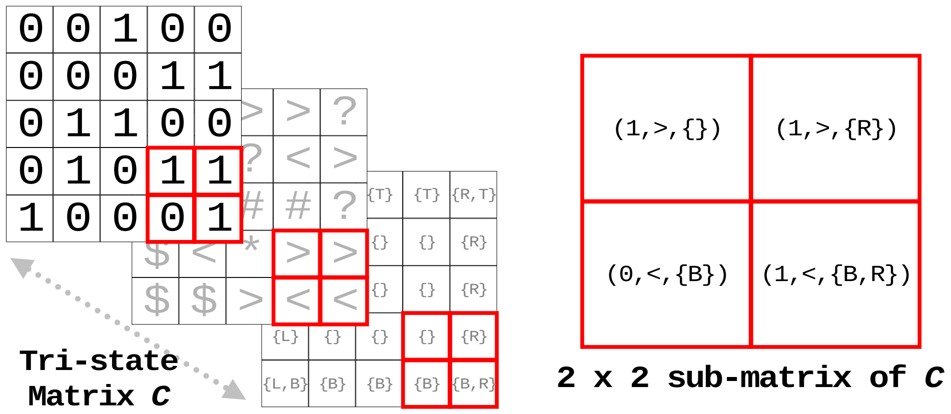


Figure 2.2   A 2 × 2 sub-matrix of the example configuration in Fig. 2.1.

this idea concrete by introducing the notion of a local detector.

A *local detector* for a pattern P is a predicate P_D on tri-state matrices of at most some fixed dimensions, such that for any configuration C, P does not hold in C iff there is at least one sub-matrix in C that satisfies P_D. Formally, given positive integer constants K_{row} and K_{col}, P_D is a local detector for P, if P_D is a predicate on tri-state matrices of size at most K_{row} and K_{col}, such that for any $m \times n$ configuration C, $P(C_V)$ does not hold if and only if there exists a $d \in \mathcal{D}^*_{m,n}(\leq K_{row}, \leq K_{col})$ such that $P_D(C(d))$ is true.

We denote a pattern P as *locally checkable* if there exists a local detector for P.

2.3.1.1 Checkerboard example

Consider a ‘checkerboard’ pattern which is informally stated as: no two adjacent cells (in the same row or same column) have the same visible state. This pattern is locally checkable because its violations are locally detectable as shown by two instances in Fig. 2.3.

2.4 Stigmergic sorting agents (stigbots)

As described in the introduction, we model the abstract connection between a stigbot’s sensors and actuators with a stimulus-response pair. Each such pair defines a *rule*, and all stigbots in a swarm have an identical set of rules called the *ruleset*. Intuitively, a stigbot’s

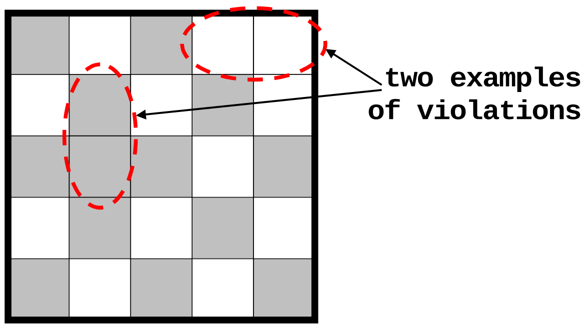


Figure 2.3 Violations of the checkerboard pattern—i.e. two adjacent cells having the same visible state—are locally detectable; each violation is detectable within a sub-matrix of size either 1×2 or 2×1.

ruleset represents the genetically programmed behavior of a social insect, i.e. the actions of a stigbot can be considered to be completely determined by its ruleset.

2.4.1 Rules and their applications

We now formalize the concept of a rule as a stimulus-response pair. The stimulus and the response of a rule, are both predicates on tri-state matrices (of same dimensions), such that any matrix matching the response contains the same number of *1* cells as any matrix matching the stimulus, and the positional state of the corresponding cells in both matrices are the same. Thus, a $p \times q$ *stigmergic rule* Rx is a pair (STx, RSx), where STx and RSx are unary predicates on $p \times q$ tri-state matrices, such that for any pair of tri-state matrices M, M' :

$$(STx(M) \wedge RSx(M')) \Longrightarrow (\#_1(M_V) = \#_1(M'_V) \ \wedge \ M_P = M'_P).$$

A *stigmergic ruleset* $\mathcal{R}$ is a set of stigmergic rules, all associated with a label set $\mathcal{L}_{\mathcal{R}}$, where $\mathcal{L}_{\mathcal{R}}$ is some fixed set restricting the labels that can used by the rules. Formally, if M is a tri-state matrix satisfying either a stimulus or response predicate of a rule in $\mathcal{R}$, then every entry of M_A belongs to $\mathcal{L}_{\mathcal{R}}$.

2.4.1.1 Rule application semantics

We define the semantics of a rule application in a configuration by describing the relationship between the pre-configuration, and the post-configuration resulting from the application.

An *application* of a $p \times q$ rule Rx in a $m \times n$ configuration C results in an $m \times n$ configuration C', if there is a rectangular domain d, where $d = \mathcal{D}^{i,j}(p,q)$ (for some $1 \leq i \leq m$, and $1 \leq j \leq n$), such that:

(1) $STx(C(d)) \wedge RSx(C'(d))$, and (2) $\forall\, 1 \leq i \leq m, 1 \leq j \leq n : (i,j) \notin d \Longrightarrow (C[i,j] = C'[i,j])$.

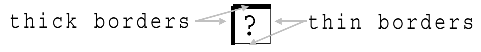
We refer to domain d in the above definition as the *site* of the application of Rx in C. The first condition says that the same group of cells must match the stimulus in C and the response in C', while the second condition requires *containment*, i.e. all the other cells outside of the application site should not be affected.

Atomicity We assume that once a stigbot starts a rule-application at some site, it does not sense any other stimuli, nor can it be interrupted by other stigbots, until the rule has been completely applied.

No overlap We assume that the behavioral characteristics of the stigbots ensures that any two concurrent application sites will be disjoint w.r.t their cells.

2.4.1.2 Graphical notation for rules

We find it easier to express our rules using a graphical notation consisting of a grid of squares to represent the possible tri-state matrices that can satisfy the stimulus/response predicates. We use no-fill squares for *1* values and gray-filled squares for *0* values of the visible state. We display the label corresponding to the auxiliary state, on top of the square. Thick borders of the square specify the cell's positional state—the cell has no neighbors to the side with the thick border. Thin borders however are not the opposite, but are instead an *underspecification* of positional state so as to avoid repeating rules. Thus a thin border does not necessarily mean that there has to be a neighbor on that side.

For example, consider the following square representation of a single cell in some configuration: 

One possible cell matching this notation is the triple: (1, { 'T','L'}, '?'). But it can also match any cell which had more edge indicators besides top and left. Thus it could also match (1, { 'T', 'L', 'B' }, '?') for example. In short it will match any cell whose positional state value is $*$, where $\{$'T','L'$\} \subseteq * \subseteq \mathcal{E}$.

2.4.2 Active stimuli locations

The set of possible sites for applications of a $p \times q$ rule Rx in a $m \times n$ configuration C, denoted by $\mathcal{SITES}_{Rx}(C)$, is the set of all sub-matrices in C that satisfy the stimulus for rule Rx: $\mathcal{SITES}_{Rx}(C) = \{d \in \mathcal{D}^*_{m,n}(p,q) : STx(C(d))\}$. We say that the stimulus for rule Rx is *active* in a configuration C, if $\mathcal{SITES}_{Rx}(C) \neq \emptyset$.

The set of all possible sites for the applications of any of the rules in a ruleset $\mathcal{R}$, in a configuration C, is denoted $\mathcal{SITES}^{\mathcal{R}}_*(C)$ and is equal to $\bigcup\{\mathcal{SITES}_{Rx}(C) : Rx$ is a rule of $\mathcal{R}\}$. Each element of $\mathcal{SITES}^{\mathcal{R}}_*(C)$ is *an active stimulus location* for $\mathcal{R}$ in C. Configuration C is *terminal* for a ruleset $\mathcal{R}$ if $\mathcal{SITES}^{\mathcal{R}}_*(C) = \emptyset$.

The left side of Fig. 2.4 shows a ruleset for sorting the CG pattern (explained further in Sec. 3.1), and the right side shows the active stimuli for this ruleset in an example configuration. Note that the active stimuli can contain overlapping sites as seen. The stigbots can choose any non-empty, non-overlapping subset of these stimuli locations for rule applications. This is explained in detail, next.

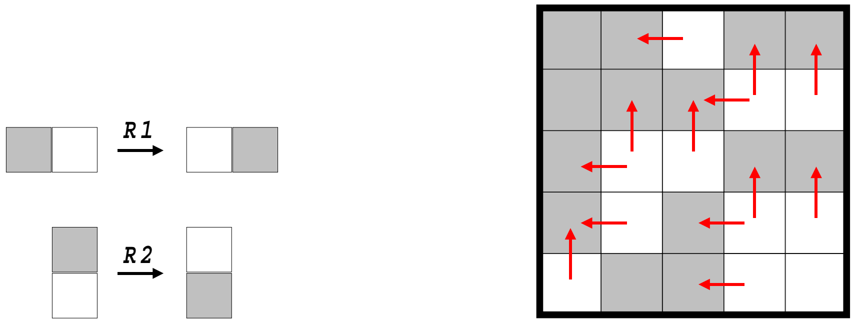


Figure 2.4 Active stimuli for the CG ruleset.On the left is the CG ruleset, while on the right are the active stimuli for this ruleset in an example configuration. The right-to-left arrows show the possible sites for application of rule $R1$, while the vertical red arrows show the sites for applications of rule $R2$.

2.4.3 Ruleset executions

An execution of a ruleset on some initial configuration consists of asynchronous and possibly concurrent rule applications at various non-deterministically chosen active stimuli locations, by a non-empty collection of stigbots. Since each rule application in an execution is triggered by its stimulus being active in the formation, the execution ends when there are no more active stimuli left. Thus, an execution of ruleset $\mathcal{R}$ must start in some configuration that is not terminal for $\mathcal{R}$, and can only end in some other configuration which is terminal for $\mathcal{R}$.

Progress We assume the following liveness property for all executions: if there are one or more active stimuli locations in a configuration, then at least one will be eventually chosen for a rule application. This assumption implies *no deadlock*, i.e. if two or more stigbots simultaneously sense either the same active stimulus or different active stimuli with overlapping sub-matrices, then they all contend for the sub-matrix matching their respective stimuli, and one stigbot emerges as the eventual winner and gets to apply its rule atomically. The progress assumption does not however mean *no starvation*, because an active stimulus location may never be chosen for rule application (i.e. will be starved) if there is always some other stimulus location active simultaneously. Thus an active stimulus location is guaranteed to have an eventual response only when there are no other active stimuli in the configuration.

Containment and no overlap assumptions together ensure that any changes being made by a rule application are not visible to any other concurrent rule applications. This independence of concurrent applications allows us to serialize any admissible execution by ordering its constituent rule applications according to their start times, breaking ties arbitrarily. Taken together with the progress and the atomicity assumptions, execution serializability aids in reasoning about correctness because, any admissible execution by a group of non-zero number of stigbots is now functionally equivalent to some admissible execution by a single stigbot carrying out rule applications in a sequential and non-deterministic manner.

2.5 Sorter definitions

We now formally define two types of stigmergic solutions for pattern sorting—namely local and non-local—by giving the conditions that a ruleset must satisfy in order for it to be considered to be of either type.

A ruleset $\mathcal{R}$ is a *stigmergic sorter* for a pattern P, if for any configuration C with a default auxiliary state matrix C_A, the following two conditions are satisfied:

1. If $\neg P(C_V)$, then C is not a terminal configuration for $\mathcal{R}$, and every execution of $\mathcal{R}$ starting from C terminates in some P-configuration.

2. If $P(C_V)$, then every execution of $\mathcal{R}$ starting from C is finite and preserves the visible matrix C_V, i.e. if C' is a configuration occurring in some execution starting from C, then $C_V = C'_V$. Thus, in this case, any execution must affect only the auxiliary state matrix.

If a ruleset $\mathcal{R}$ is a stigmergic sorter for a pattern P and if the rules of $\mathcal{R}$ do not change the auxiliary states of cells, i.e. $\mathcal{L}_{\mathcal{R}} = \{\text{` '}\}$, then $\mathcal{R}$ is a *local stigmergic sorter* for P.

A pattern P is *(locally) stigmergic-sortable* iff there exists a (local) stigmergic sorter for P.

2.5.1 Local checkability is necessary for local sorting.

Based on the definitions of locally checkable patterns and local stigmergic sorters, it is straightforward to show that local checkability is a necessary condition for a pattern to be locally sorted. The idea is that the disjunction of all the stimulus predicates of a ruleset that is a local stigmergic sorter for a pattern, constitutes a local detector for that pattern. We state this observation as a lemma for future reference.

Lemma 1 *A pattern P is locally stigmergic-sortable only if P is locally checkable.*

Proof: Let P be a locally sortable pattern, and $\mathcal{R}$ be some ruleset which can locally sort P, i.e. without using labels. For any configuration C, if C does not satisfy P, then by condition 1 in the above definition of stigmergic sorters, C must contain at least one active stimulus for some rule of $\mathcal{R}$. On the other hand if C does satisfy P, then by condition 2 of the above definition—and because $\mathcal{R}$ does not use labels—C cannot contain any active stimulus for the rules of $\mathcal{R}$. The logical disjunction of all the stimulus predicates for the rules of $\mathcal{R}$ thus forms a local detector for sub-matrices of size at most $maxrow_{\mathcal{R}} \times maxcol_{\mathcal{R}}$, where $maxrow_{\mathcal{R}}$ and $maxcol_{\mathcal{R}}$ are respectively the largest row and column spans in the stimulus predicates for the rules of $\mathcal{R}$. It follows that P is locally checkable. Thus P is locally stigmergic-sortable $\Longrightarrow P$ is locally checkable. ■

2.5.2 Transforms and legality of configurations w.r.t. rulesets

Given a ruleset $\mathcal{R}$, a configuration C' is a $\mathcal{R}$-$transform$ of a configuration C if there exists a (possibly empty) sequence of rule-applications drawn from $\mathcal{R}$, that transforms C into C'. Clearly, the $\mathcal{R}$-$transform$ relation is reflexive and transitive.

Since the definition of sorters given above requires that sorting rulesets work correctly only on input configurations that have a default auxiliary state matrix, we can formalize the legality of a configuration w.r.t. a ruleset as given below.

A configuration C is *legal* for a ruleset $\mathcal{R}$ if and only if C is a $\mathcal{R}$-$transform$ of some configuration with a default auxiliary state matrix.

CHAPTER 3. Three pattern paradigms

We now consider three instances of pattern sorting, each from a general pattern paradigm. The three paradigms are informal pattern classifications which we intuitively feel might be useful as initial targets of experimentation. Although the specific pattern instances we cover are of course binary, these general paradigms can also apply to patterns of cells with any number of different visible states (not just binary). Our three paradigms (in order of their instances covered here) are (1) *clustering* which includes patterns in which cells with the same visible state are grouped together in a single cluster, (2) *axial separation*, covering patterns in which cells with different visible states are cleanly separated along some axis, and (3) *connectivity* which refers to patterns in which cells having some particular visible state are all connected together.

3.1 Corner Grouped pattern (CG)

This pattern specifies that *1* cells in any row must be to the left of any *0* cells in that row, and *1* cells in a column must be above any *0* cells in the same column. Roughly speaking, the *1*s are grouped around top left corner, while the *0*s are grouped around the bottom right corner. I.e., the pattern predicate CG is defined as follows. For any $m \times n$ configuration C, $CG(C_V)$ holds iff $\forall\ 1 \leq i, i' \leq m; 1 \leq j, j' \leq n$:

$$(C_V[i,j] \wedge \neg C_V[i',j']) \Longrightarrow (i < i' \vee j < j').$$

Figure 3.1 shows the visible states of three example configurations—the first two satisfy CG, while the last one does not because there are *0* values above and to the left of some *1* values.

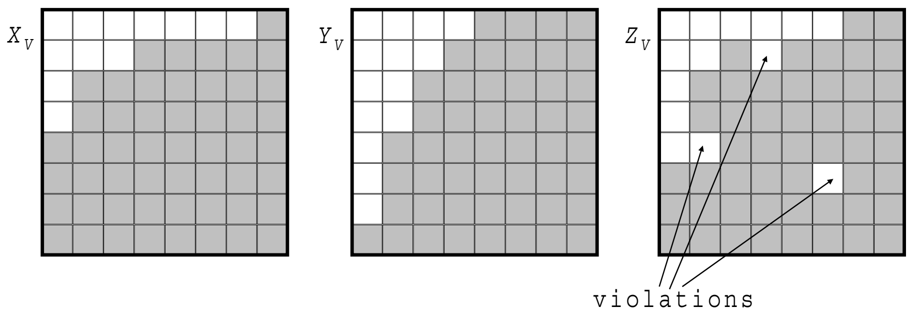


Figure 3.1 Visible state matrices of three 8×8 configurations called X, Y and Z. While X and Y are CG-configurations, Z is not a CG-configuration—the arrows show the violations in Z_V (white square = *1*-cell and gray square = *0*-cell).

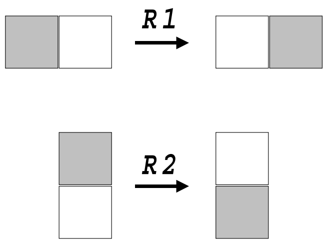


Figure 3.2 A 2-rule ruleset that is a local stigmergic sorter for CG. No re-labeling happens in either rule i.e. $\mathcal{L}_{\mathcal{R}} = \{\text{' '}\}$.

3.1.1 A local stigmergic sorter

Figure 3.2 shows a simple ruleset—consisting of just two swapping rules. Rule $R1$ swaps the visible states between two horizontally adjacent cells, while rule $R2$ swaps them in two vertically adjacent cells. We now outline the proof that this ruleset is a local stigmergic sorter for CG; we go into some detail in order to illustrate the use of a classical distributed computing idea—variant functions—in a stigmergic context.

3.1.1.1 Cost function

Variant functions [4] are used to show convergence in self-stabilizing systems. The idea is to assign a cost function over the configuration set, whose value is bounded and then show that (a) the cost monotonically decreases with each individual step in the stabilizing algorithm, and

(b) after the cost reaches a certain threshold the system has entered a stable configuration.

We use this idea in the stigmergic context by defining a simple cost function that measures the amount of 'disorder' w.r.t to pattern CG in each configuration. Then we show that each of the two rule applications in Fig. 3.2 reduce this disorder, thus monotonically reducing the cost to zero, at which point CG is satisfied. We define the cost function as follows: the total cost for a $m \times n$ configuration is the sum of costs for each individual cell in the configuration; for any *1*-cell the cost is zero, while for each *0*-cell (i, j), the cost is the number of *1*-cells in the rectangle whose top-left corner is (i, j) and bottom-right corner is (m, n). We will refer to this rectangle as the *cost rectangle* of cell (i, j). Figure 3.3 shows an application of this cost function in calculating the individual costs of three *0*-cells in a configuration example.

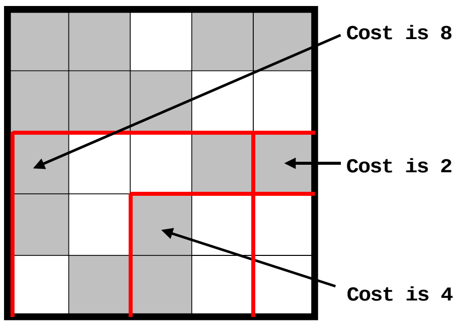


Figure 3.3 The cost of a *0*-cell is just the number of he number of *1*-cells within its cost rectangle. The cost rectangles for each of the three example *0*-cells are shown in red.

Formally, the cost function is as follows: For a $m \times n$ configuration C, $totalCost(C) = \sum_{i=1}^{m}\sum_{j=1}^{n} cost(C_V, i, j)$,

$$\text{where } cost(C_V, i, j) = \begin{cases} 0 & \textit{if } C_V[i,j] \\ \displaystyle\sum_{i'=i}^{m}\sum_{j'=j}^{n} C_V[i',j'] & \textit{if } \neg C_V[i,j] \end{cases}$$

We now prove some useful lemmas relating the cost function to the ruleset of Fig. 3.2.

Lemma 2 *For any configuration C, totalCost$(C) \geq 0$.*

Proof: This follows from the fact that the total cost of a configuration is a summation of the non-negative cost of each individual cell. ■

Lemma 3 *For any configuration C, totalCost$(C) = 0$ iff C is a CG-configuration.*

Proof: For a configuration C, *totalCost*$(C) = 0$

$\iff$ the cost of each *0*-cell in C is zero,

$\iff$ the cost rectangle of each *0*-cell contains no *1*-cells,

$\iff$ every *1*-cell is above or to the left of each *0*-cell,

$\iff$ the configuration satisfies CG. ■

Lemma 4 *For any configuration C,*

$\mathcal{SITES}_{R1}(C) \cup \mathcal{SITES}_{R2}(C) \neq \emptyset \iff \neg CG(C_V)$.

Proof: For a $m \times n$ configuration C,

$\mathcal{SITES}_{R1}(C) \cup \mathcal{SITES}_{R2}(C) \neq \emptyset$,

$\iff$ a *1*-cell is directly to the right of a *0*-cell in the same row
or a *1*-cell is directly below a *0*-cell in the same column,

$\iff$ configuration C does not satisfy CG.

$\iff$ $\neg CG(C_V)$. ■

Lemma 5 *If an application of rule $R1$ or $R2$ in some configuration C results in a configuration C', then totalCost$(C') <$ totalCost(C).*

Proof: We will prove the claim only for an application of rule $R1$; the argument for an application of $R2$ has the exact same structure. Let (i, j) and $(i, j + 1)$ be the two cells in a 1×2 rectangular domain that is the site of application for $R1$ in C. Clearly, $C_V[i, j] = 0$ and $C_V[i, j + 1] = 1$ (to meet the stimulus specification of $R1$). We now consider the costs of cells (i, j) and $(i, j + 1)$, before and after the rule application.

Before application of $R1$ (refer to Fig. 3.4): Since $C_V[i,j] = 1$, by the cost function $cost(C_V, i, j+1) = 0$. We assume $cost(C_V, i, j) = x$, i.e. the cost rectangle of cell (i,j) in C contains x number of *1*-cells (including cell $(i, j+1)$).

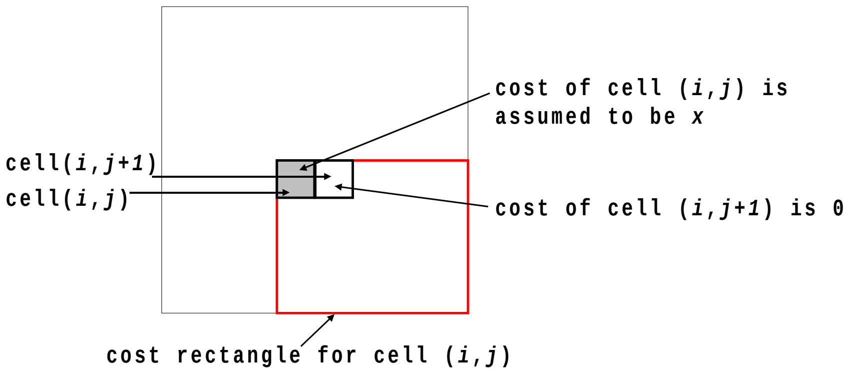

cost of cell (i,j) is assumed to be x
cell(i,j+1)
cell(i,j)
cost of cell (i,j+1) is 0
cost rectangle for cell (i,j)

Figure 3.4 The costs of cells (i,j) and $(i,j+1)$ in C, before the application of rule $R1$.

After application of $R2$ (refer to Fig. 3.5): Now $cost(C'_V, i, j) = 0$, while $cost(C'_V, i, j+1) = x - 1 - \sum_{i'=i+1}^{m} C'_V[i', j]$. The former follows from the cost function, while the latter can be informally explained as follows: the cost rectangle of cell $(i, j+1)$ in C' is formed by excluding the cell $(i, j+1)$ and the region D (see Fig. 3.5) from the cost rectangle of (i,j) in C (the summation term $\sum_{i'=i+1}^{m} C'_V[i', j]$ in the above cost equation is just the number of *1*-cells in region D).

By comparing the costs before and after the application, it is clear that the total cost of cells (i,j) and $(i, j+1)$ is at least one less in C' than in C. To complete the proof, we will now show that the total cost of all the other cells is non-increasing. For this we partition the rest of the cells into three regions A, B, and C as shown in Fig. 3.6. Formally, the three regions can be described as follows:

$$\mathbf{A} = \{(i', j') : 1 \le i' \le i \ \wedge \ 1 \le j' \le j\} - (i,j)$$

$$\mathbf{B} = \{(i', j') : 1 \le i' < i \ \wedge \ j' = j+1\}$$

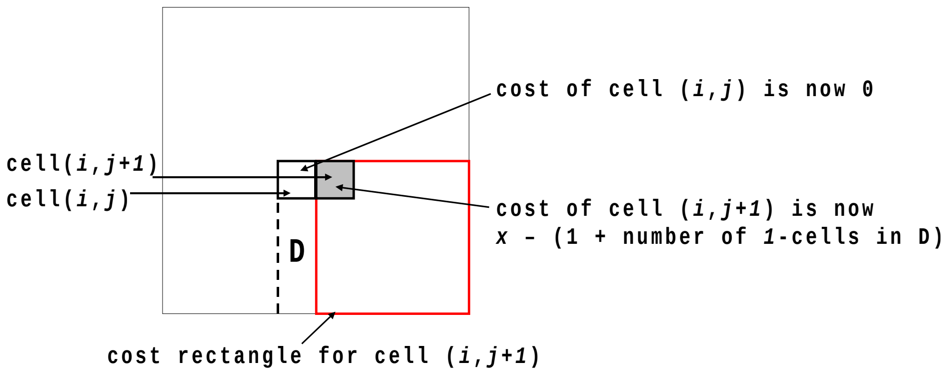

cost of cell (*i*,*j*) is now 0
cell(*i*,*j*+*1*)
cell(*i*,*j*)
cost of cell (*i*,*j*+*1*) is now
x - (1 + number of *1*-cells in D)
D
cost rectangle for cell (*i*,*j*+*1*)

Figure 3.5 The costs of cells (i, j) and $(i, j+1)$—in configuration C'—after the the application of rule $R1$.

C $= \{(i', j') : i' > i \ \vee \ j' > j+1\}$

Based on the formal descriptions of the cost function and the above three regions, it is straightforward to derive the following relationships between the cells (i, j) and $(i, j+1)$, and the cost rectangles for any *0*-cells in each of the three regions (see Fig. 3.6):

A: the cost rectangles include both cells (i, j) and $(i, j+1)$.

B: the cost rectangles include the cell $(i, j+1)$, but not (i, j).

C: the cost rectangles exclude both cells (i, j) and $(i, j+1)$.

From the above inclusion-exclusion relationships it is easy to see that the cost of each *0*-cell in both region A and region C is the same in configuration C as well as C', while the cost of each *0*-cell in region B is reduced by 1 in C' (due to cell $(i, j+1)$ becoming a *0*-cell), see Fig. 3.7. The cost of a *1*-cell in any region, of course, remains zero in either configuration. Thus the total cost of all the other cells besides (i, j) and $(i, j+1)$ is non-increasing.

■

Using the above lemmas we now prove the correctness of the local sorter for CG.

Theorem 6 *The ruleset given in Fig. 3.2 is a local stigmergic sorter for CG.*

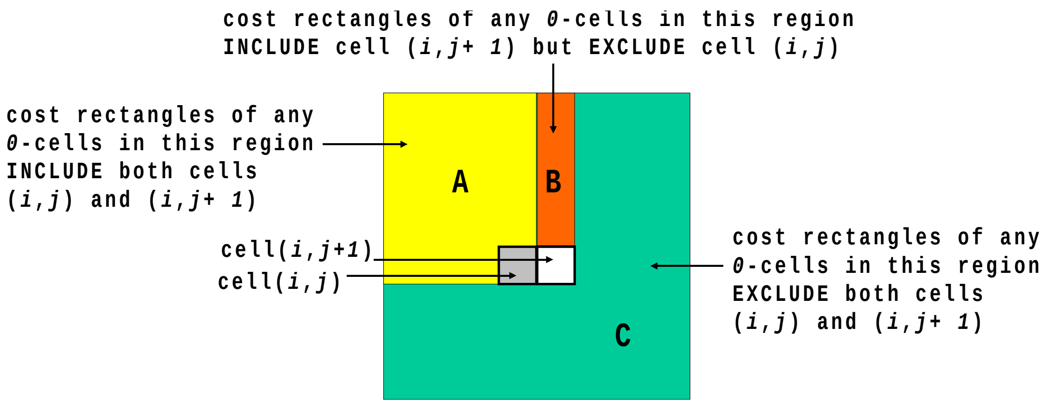

cost rectangles of any 0-cells in this region INCLUDE cell (i,j+ 1) but EXCLUDE cell (i,j)
cost rectangles of any 0-cells in this region INCLUDE both cells (i,j) and (i,j+ 1)
A
B
cell(i,j+1)
cell(i,j)
cost rectangles of any 0-cells in this region EXCLUDE both cells (i,j) and (i,j+ 1)
C

Figure 3.6 Partitioning the remaining cells (i.e. those besides (i, j) and $(i, j + 1)$) into regions A, B and C.

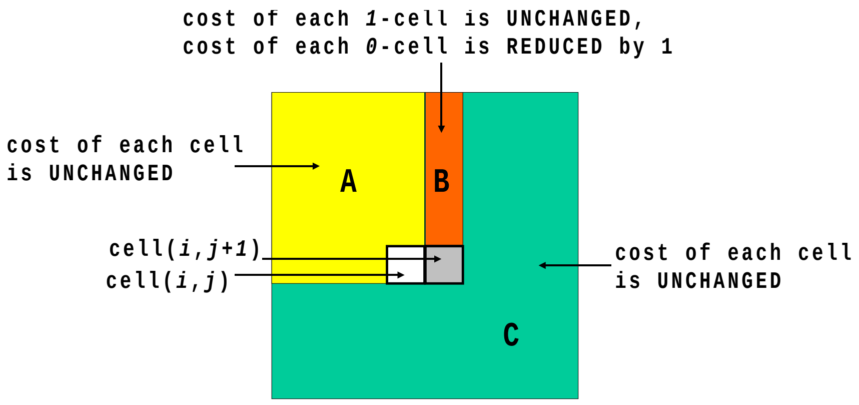

cost of each 1-cell is UNCHANGED, cost of each 0-cell is REDUCED by 1
cost of each cell is UNCHANGED
A
B
cell(i,j+1)
cell(i,j)
cost of each cell is UNCHANGED
C

Figure 3.7 Effect of the application of $R1$ on the costs of each of the cells in the regions A, B and C.

Proof: We first show that the ruleset in Fig. 3.2 satisfies the two conditions in the definition of stigmergic sorter given in Sec. 2.5. For condition 1, let C be some non CG-configuration. By Lemma 4, C is not a terminal configuration for the ruleset. By Lemma 5, each successive configuration resulting from consecutive rule applications in any non-empty (serialized) execution has a strictly lower total cost than its predecessor. Since Lemma 2 tells us that this cost cannot decrease indefinitely, therefore each execution starting from C must eventually terminate. Due to Lemmas 3 and 4, none of these executions can terminate in configurations with total cost > 0. Thus each execution starting from C terminates with a configuration whose total cost

= 0, i.e. a CG-configuration, and so condition 1 is satisfied. Condition 2 is trivially satisfied because, by Lemma 4, if C is a CG-configuration, then there are no executions starting from C. Thus the ruleset of Fig. 3.2 meets the definition of a stigmergic sorter for CG. Since neither of its two rules use relabeling, therefore, this ruleset is also a local stigmergic sorter for CG.

■

Corollary 7 *CG is locally stigermic-sortable.*

3.2 Two Layered pattern (TL)

In this pattern *1* and *0* cells form two horizontal layers in the grid, with the *1*-cells layer on top. A possibly mixed row of *1*s and *0*s may separate the two layers, but all the *1*s in this mixed row must be to the left of all the *0*s in the row. For a $m \times n$ configuration C, $TL(C_V)$ holds iff $\forall\ 1 \leq i, i' \leq m; 1 \leq j, j' \leq n$:

$$(C_V[i,j] \wedge \neg C_V[i',j']) \implies (i < i' \vee (i = i' \wedge j < j'))$$

For example, in Figure 3.8, X and Y satisfy the TL pattern specification, but Z does not.

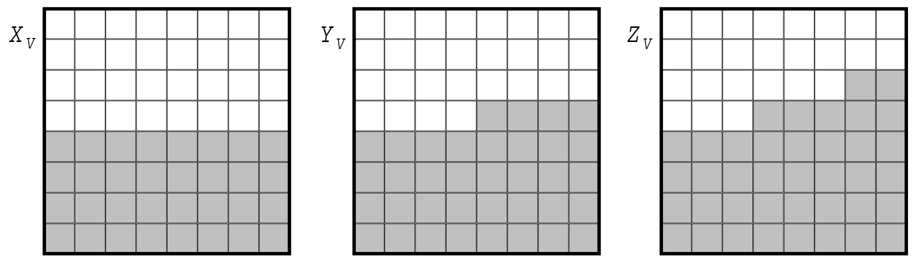


Figure 3.8 The visible states of three 8×8 configurations called X, Y and Z. While X and Y are TL-configurations, Z is not a TL-configuration, because there can be at most one row whose cells do not all have the same visible state.

Theorem 8 *TL is not locally checkable.*

Proof: We prove the result by contradiction. Assume, that pattern TL is locally checkable. Therefore, there must exist positive integers K_{row} and K_{col}, and a detector predicate TL_D, such that for any configuration C, C is not a TL-configuration if and only if there is a sub-matrix in C with dimensions at most K_{row} by K_{col} that satisfies the detector predicate TL_D.

Given constants K_{row} and K_{col}, we construct a non TL-configuration C, and two TL-configurations C' and C'', each with 2 rows and $K_{col}+1$ columns, such that any sub-matrix of C is equivalent to some sub-matrix of either C' or C'', and therefore cannot satisfy the detector predicate TL_D, giving a contradiction. We define C, C' and C'' below. Formally, we prove that for all rectangular domains d in $\mathcal{D}^*_{2,K_{col}+1}(\leq K_{row}, \leq K_{col})$, $C(d)$ cannot satisfy predicate TL_D.(Recall that $\mathcal{D}^*_{2,K_{col}+1}(\leq K_{row}, \leq K_{col})$ denotes all rectangular domains of size at most $K_{row} \times K_{col}$ within a $2 \times K_{col}+1$ matrix.)

We define configuration C (see Fig. 3.9) with 2 rows and $K_{col}+1$ columns, with a default auxiliary state matrix, and a visible state matrix as follows: (first row) $\forall\ 1 \leq j \leq K_{col}$: $C_V[1,j] \wedge \neg C_V[1, K_{col}+1]$, and (second row) $C_V[2,1] \wedge (\forall\ 2 \leq j \leq K_{col}+1 : \neg C_V[2,j])$. Next, we define two TL-configurations C' and C'' (see Fig. 3.9) such that each differs from C in the visible state of exactly one cell: C' is exactly the same as C except $C'_V[1, K_{col}+1]$, and C'' is exactly the same as C except $\neg C''_V[2,1]$.

Next, we group the domains of $\mathcal{D}^*_{2,K_{col}+1}(\leq K_{row}, \leq K_{col})$ into two proper (but not disjoint) subsets; $\mathcal{S}_1$ are the domains which exclude $(1, K_{col}+1)$ and $\mathcal{S}_2$ are those which exclude $(2,1)$. Thus $\mathcal{S}_1 = \{d : (1, K_{col}+1) \notin d\}$, and $\mathcal{S}_2 = \{d : (2,1) \notin d\}$. Due to the adversarial choice for the number of columns to be $K_{col}+1$, there is no domain in $\mathcal{D}^*_{2,K_{col}+1}(\leq K_{row}, \leq K_{col})$ which contains both of the cells $(2,1)$ and$(1, K_{col}+1)$. Hence $\mathcal{S}_1 \cup \mathcal{S}_2 = \mathcal{D}^*_{2,K_{col}+1}(\leq K_{row}, \leq K_{col})$.

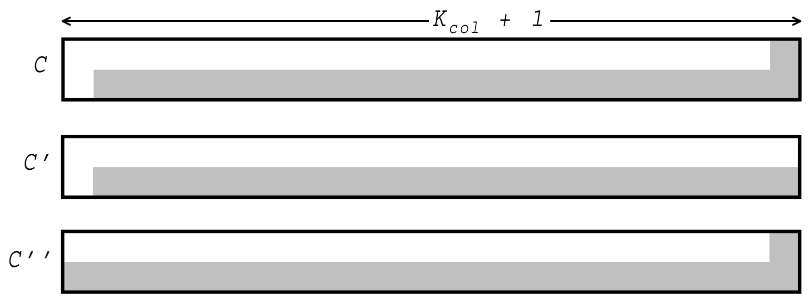


Figure 3.9 C, C' and C'' are each $2 \times (K_{col}+1)$ configurations. C is not a TL-configuration, while both C' and C''—each differing from C by the visible state of just one cell—are both TL-configurations.

Now we are set up to exploit the indistinguishability of the three configurations defined above since they contain equivalent sub-matrices. Consider C and C': since the domains in $\mathcal{S}_1$ do not include cell $(1, K_{col}+1)$, therefore every domain in $\mathcal{S}_1$ evaluates to the same sub-matrix in both C and C'. I.e., $\forall\ d \in \mathcal{S}_1 : C(d) = C'(d)$. Arguing similarly for C, C'' and the domains of $\mathcal{S}_2$, we have $\forall\ d \in \mathcal{S}_2 : C(d) = C''(d)$. Thus, for every sub-matrix $C(d)$ in C—of size at most $K_{row} \times K_{col}$—there is a TL-configuration that has the same exact sub-matrix. Now, by definition of a non-TL configuration, there exists some sub-matrix $C(d)$ that satisfies TL_D. Since there exists a TL-configuration E such that $C(d) = E(d)$, this implies that $E(d)$ satisfies TL_D, contradicting the fact that E is a TL-configuration.

■

Corollary 9 *TL is not locally stigermic-sortable.*

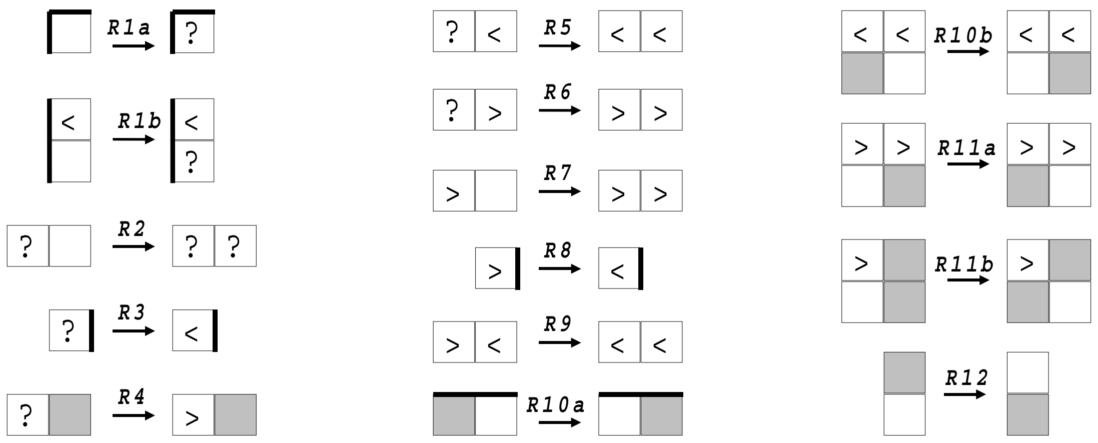


Figure 3.10 A 15-rule ruleset $\mathcal{R}$ that is a (non-local) stigmergic sorter for TL, with $\mathcal{L}_{\mathcal{R}} = \{$' ','?',' >',' <'$\}$.

3.2.1 A non-local stigmergic sorter

Fig. 3.10 shows a 4-label ruleset for stigmergically sorting TL. Figure 3.11 shows a simple execution example of this ruleset. As can be seen in the execution, an invariant of this ruleset is that once a cell acquires a non-default label, its visible state does not change again for the

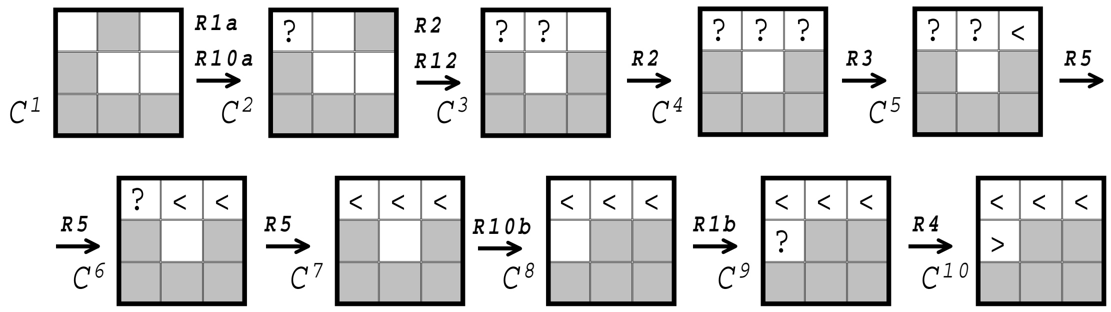


Figure 3.11 An example execution of $\mathcal{R}$ starting from configuration C^1, and ending in configuration C^{10}. Several transitions have concurrent rule applications.

rest of the execution. The '?' label is like a query propagated rightwards to check if the current row is completely filled with *1*-cells. The '>' and '<' labels act as directional signs for the stigbots telling them which way to swap the visible states of cells in the row directly beneath.

We sketch an outline of the correctness proof for the the ruleset in Fig. 3.10, using the following cost function.

3.2.1.1 Cost function

For a $m \times n$ configuration C, $totalCost(C) = \sum_{i=1}^{m}\sum_{j=1}^{n} cost(C,i,j)$, where

$$cost(C,i,j) = \begin{cases} \sum_{i'=i+1}^{m}\sum_{j'=1}^{n} C_V[i',j'] & \textit{if } \neg C_V[i,j] \\ \\ Acost(C,i,j) & \textit{if } C_V[i,j] \end{cases}$$

$$Acost(C,i,j) = \begin{cases} 4 & \textit{if } C_A[i,j] = \text{` '} \\ 3 & \textit{if } C_A[i,j] = \text{`?'} \\ Acost^{>}(C,i,j) & \textit{if } C_A[i,j] = \text{`>'} \\ Acost^{<}(C,i,j) & \textit{if } C_A[i,j] = \text{`<'} \end{cases}$$

$$Acost^{>}(C,i,j) = \begin{cases} 2 & \textit{if } (\forall\ 1 \le j' \le n : C_V[i,j'])\ \vee\ \neg(\forall\ i < i' \le m; 1 \le j' \le n : \neg C_V[i,j']) \\ \\ 0 & \text{otherwise} \end{cases}$$

$$Acost^{<}(C,i,j) = \begin{cases} 1 & \textit{if } \neg(\forall\ 1 \le j' \le n : C_V[i,j']) \\ \\ 0 & \text{otherwise} \end{cases}$$

A brief explanation of the cost function The total cost of the configuration is the sum of individual cost of each cell in the configuration given by the *cost*() function. This function assigns the cost of a *0*-cell as being the number of *1*-cells that are below it[1]. For a *1*-cell this function uses the *Acost*() function to compute the cost based on the label of the *1*-cell. The *Acost*() function assigns *1*-cells with a default label and a '?' label, the costs of 4 and 3 respectively. For *1*-cells with labels '>' and '<' it uses the $Acost^{>}()$ and $Acost^{<}()$ functions respectively to assign costs. The $Acost^{>}()$ function assigns a *1*-cell with label '>' the cost of 2 if the cell is in a row that contains all *1*-cells or if there is at least one *1*-cell in the rows below. If neither of these two conditions hold, then the cost of that *1*-cell with label '>' is 0. The $Acost^{<}()$ function assigns a *1*-cell with label '<' the cost of 1 if there is at least one *0*-cell in its row. If not, then its cost is 0.

We will show the correctness of the ruleset of Fig. 3.10 by connecting the changes in the total cost of a configuration—as given by the above cost function—with the application of the rules from the ruleset of Fig. 3.10. We give semi-formal proofs and proof outlines for the subsequent Lemmas and Theorem; a formal and completely rigorous proof of correctness would probably be quite tedious and lengthy. We use the following in Lemmas 10-17:

- $\mathcal{R}$ is the ruleset of Fig. 3.10.
- C is any $m \times n$ configuration which is legal for $\mathcal{R}$ (see Sec. 2.5.2 for legality).

[1] The TL specification requires not only that there should be no *1*-cell below a *0*-cell, but also that in a mixed row, if any, the *1*-cells should all be to the left of the *0*-cells. We ignore this latter requirement in the cost function because doing so allows the application of any rule from the ruleset of Fig. 3.10 to have a non-increasing effect on the total cost of the configuration. This simplifies reasoning about the executions of the ruleset.

Lemma 10 *A cell in C has a non-default label only if this cell and all cells to the left of it in the same row are* 1*-cells, i.e.* $\forall\ 1 \leq i \leq m, 1 \leq j \leq n : C_A[i,j] \neq$ ' ' $\Longrightarrow \forall\ 1 \leq j' \leq j : C_V[i,j]$

Proof: A cell with a default label gets a non-default label only via applications of rules $R1a$, $R1b$, $R2$, and $R7$. The stimulus predicates of these four rules each require that the cell be a *1*-cell, and that either the cell be the leftmost in the row (rules $R1a$ and $R1b$) or already have a *1*-cell with a non-default label as its left neighbor ($R2$ and $R7$). Finally, the rules of $\mathcal{R}$ do not ever change a cell's non-default label back to the default, and they also do not change the visible state of a cell once it has a non-default label, thus proving the claim.

■

Lemma 11 *The total cost of C is non-negative, i.e. $totalCost(C) \geq 0$*

Proof: This follows from the fact that the total cost of a configuration is a summation of the non-negative cost of each individual cell. ■

Lemma 12 *The total cost of C is zero only if C is a TL-configuration, i.e. $totalCost(C) = 0 \Longrightarrow TL(C_V)$*

Proof: We argue this by constructing the different types of configurations that have zero total costs, and then filtering out only those that are legal configurations.

The cost function stipulates that for any configuration to have a zero total cost, the individual cost of each cell in the configuration must be also zero, i.e. $cost(C,i,j) = 0$. This gives us the following constraints on such configurations:

1. No *0*-cell must ever be above a *1*-cell. This means that the *1* and *0* cells in the configuration are separated into two vertical layers with a possibly mixed row of cells in between the two layers.

2. The *Acost*() for each *1*-cell must be 0. This gives us the following sub-constraints on the labels of *1*-cells.

   (a) Each *1*-cell in a row of all *1*-cells must have the '<' label (by $Acost^{<}()$).

(b) Each *1*-cell in the mixed row (if any) must have the '>' label (by $Acost^{>}()$).

These constraints give us three different types of configurations that have a zero total cost. These three types are shown in Fig. 3.12. Lemma 10 forbids any *0*-cell to be to the left of any *1*-cell with a non-default label. Thus configuration T_X of Fig. 3.12 is clearly not legal for $\mathcal{R}$ because of its violations of Lemma 10 in the disordered mixed row. It can be easily seen from Fig. 3.12 that both T_1 and T_2 are TL-configurations, thus proving the claim. Note that we need not prove the legality of either T_1 or T_2 w.r.t. ruleset $\mathcal{R}$.

■

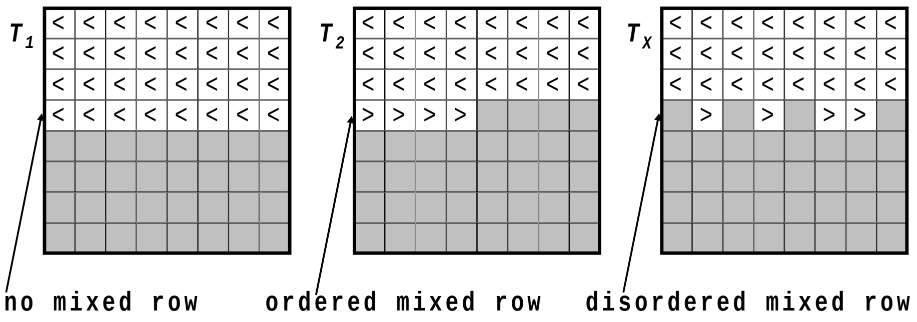


Figure 3.12 T_1, T_2, and T_X are the three possible types of configurations that have zero total cost. T_1 has no mixed row of cells, but instead has two clearly differentiated layers of *1* and *0*-cells. T_2 has a mixed row where the *1*-cells in the row are all to the left of the *0* cells in the row. T_X has a mixed row with no ordering between the *1* and *0* cells. By Lemma 10 configuration T_X is not legal for $\mathcal{R}$

Lemma 13 *C does not contain any active stimuli for $\mathcal{R}$ only if its total cost is zero, i.e.* $\mathcal{SITES}^{\mathcal{R}}_{*}(C) = \emptyset \Longrightarrow totalCost(C) = 0$

Proof: (A rough outline) We will inductively construct the different possible types of configurations which have no active stimuli for $\mathcal{R}$, i.e. terminal configurations for $\mathcal{R}$. The following argument is only a rough sketch and an actual formal proof (by exhaustive enumeration) would be much longer.

Assume that C is terminal for $\mathcal{R}$. Consider the *1*-cells of the first row of C. All the *1*-cells of the remaining rows in C are below these *1*-cells of the first row, because otherwise there would be an active stimulus for rule $R12$. Also, because of the absence of stimuli for rule $R10a$, all the *1*-cells in the first row are to the left of any *0*-cells in that row.

There are now two possibilities for visible state of the cells in the first row:

Case 1: There is at least one 0*-cell in the row.* In this case, what are possible values for the label on the leftmost cell in the first row, i.e. the corner cell? By absence of stimuli for $R1a$ it definitely cannot be the default, but could '?' be a possibility?. Consider the right neighbor of this corner cell. If the right neighbor is an *0*-cell then clearly the corner cell should be labeled '>', otherwise we have stimuli for $R4$. But what if the right neighbor of the corner cell is an *1*-cell? Then the right neighbor too cannot have a default label, otherwise we have stimuli for $R2$. Could it again have '?' as its label? For that we consider its right-neighbor and so on, until at some point the right-neighbor has to be an *0*-cell. Working backwards from there by considering the absence of stimuli for $R6$, we see that the only possibility allowed is for each *1*-cell in the row to have '>' as its label. By exhaustively considering all other cases of possible legal label assignments (such as those that involve the absence of stimuli for $R7$), it can be shown that for this case, each *1*-cell in the row having '>' as its label is the only possible result which has no active stimuli. Finally, by the absence of stimuli for rules $R11a$, $R11b$, and $R12$, there cannot be any *1*-cells in the rows below, and we are done.

Case 2: All the cells in the row are 1*-cells.* By exhaustively enumerating (and eliminating) the various label possibilities by considering with the absence of stimuli for various rules including $R3$, $R5$, $R8$ and $R9$, we can show that in this case, each *1*-cell in the row having '<' as its label is the only one which has no active stimuli. In this case there can be *1*-cells in the row directly below, however, due to the absence of rule $R10b$ all the *1*-cells in the that row are to the left of any *0*-cells in that row. The entire argument is now repeated for this row, this time using the absence of stimuli for $R1b$ instead of $R1a$. This process will terminate either with a case 1 row, or if there are no more rows.

It is easy to see that the above inductive construction will produce a terminal configuration that belongs to type T_1 or T_2 of Fig. 3.12. As argued earlier, both of these configuration types have zero total cost.

■

Lemma 14 *Let $\mathcal{R}'$ be a ruleset consisting only of the four rules: $R10a, R10b, R11a$, and $R11b$. For any configuration C' that is a $\mathcal{R}'$-$transform$ of C, $totalCost(C') = totalCost(C)$.*

Proof: None of the four rules change the labels of any cell. They simply swap the visible state of cells sideways in a row. It is easy to see from the cost function that these sideways swaps without label changes will not change the total cost. ■

Lemma 15 *There are only a finitely many different configurations that are $\mathcal{R}'$-$transforms$ of C (see Sec. 2.5.2 for transforms), i.e. all executions of $\mathcal{R}'$ starting from C are of finite length.*

Proof: The four rules swap the visible state sideways in one particular direction; rules $R10a$ and $R10b$ swap the *1*-state to the left in a row, while $R11a$ and $R11b$ swap it to the right. The stimulus predicates of these rules ensure that the visible state of a cell cannot be involved in swaps in both directions. Therefore, for each row, there can only be at most n applications of any of these rules. Thus all executions of $\mathcal{R}'$ starting from C will be of finite length. ■

Lemma 16 *Let $\mathcal{R}''$ be a ruleset consisting of all rules of $\mathcal{R}$, except for these four rules: $R10a, R10b, R11a$, and $R11b$. For any configuration C'' that is a $\mathcal{R}''$-$transform$ of C, if $C'' \neq C$ then $totalCost(C'') < totalCost(C)$.*

Proof: Using the cost function it is easy to see that, other than $R8$ and $R9$, the application of any other rules of $\mathcal{R}''$ will result in a configuration with a lesser total cost. Applications of $R8$ or $R9$ also reduce the total cost, but the argument for that are slightly longer as given below.

The reason that the argument for $R8$ and $R9$ is slightly trickier is because both of these rule applications involve changing the label of a cell from '>' to '<'. Unlike for other rules of $\mathcal{R}''$, this label change by itself is not sufficient to show that the individual cost of the cell undergoing the change is actually reduced. For that we need to also show that the cost of the cell with '>' was non-zero before application, and becomes zero after application i.e. with the new label of '<'.

To show this, first note that none of the rules of $\mathcal{R}''$ (or for that matter even $\mathcal{R}$) change the label of a cell once it has the '<' label. This fact along with the dependencies between rules $R3$, $R9$, $R8$ and $R5$ ensures that when a cell has the '<' label, then that cell and all the cells to its right up to the end of the row are *1*-cells with label '<'. This means the cell with the label '>' matching the stimulus predicate of either $R8$ or $R9$ is in a row that has all *1*-cells. By the cost function its cost is therefore 2 before the application of either of these rules. After the application that same cell has a '<' label and is still in a row with all *1*-cells. By the cost function its cost is now 0, thus proving the claim. ■

Lemma 17 *Let C be a $\mathcal{R}$-transform of a configuration C^0 with a default auxiliary state matrix. If C^0 is already a TL-configuration, then C does not contain active stimuli for any of the following five rules: $R10a, R10b, R11a, R11b$ and $R12$ i.e. $TL(C^0_V) \Longrightarrow \mathcal{SITES}_{R10a}(C) \cup \mathcal{SITES}_{R10b}(C) \cup \mathcal{SITES}_{R11a}(C) \cup \mathcal{SITES}_{R11b}(C) \cup \mathcal{SITES}_{R12}(C) = \emptyset$*

Proof: Because C^0 was a TL-configuration to begin with, and since there are no rules that can move *1*-cells downward, therefore, the visible state of all the *1*-cells and *0*-cells that are not in the mixed row of C^0 (if any) will be preserved in any execution of $\mathcal{R}$ starting from C^0 (obviously including those that lead to C). Thus it follows that C does not contain a stimulus for rule $R12$ since no *1*-cell is below a *0*-cell in C.

We now eliminate the possibility of any active stimuli for either rules $R11a$ or $R11b$ in C. An active stimulus for rule $R11a$ or $R11b$ can exist in C only if cells in the row directly above the mixed row have the '>' label. But none of the cells of this row can ever acquire the '>' label as discussed below.

A cell can get the '>' label only by previous applications of rules $R4$ or $R7$. From the dependencies between $R4$ and $R7$ it is clear that the application of $R7$ in any row needs to be preceded by an application of $R4$ in the same row. From the stimulus predicate of $R4$ it follows that a row will contain an active stimulus for $R4$ only if that row also has a *0*-cell. But since the row directly above the mixed row always has only *1*-cells (throughout any execution of $\mathcal{R}$ starting from C^0), therefore none of cells in that row will ever have the '>' label. This eliminates the possibility of stimuli for rules $R11a$ and $R11b$ from C. We now use this to argue against the existence of stimuli for $R10a$ and $R10b$ in C, as given below.

Note that because C is assumed to be any $\mathcal{R}$-$transform$ of C^0, therefore, from the discussion above, no $\mathcal{R}$-$transform$ of C^0 ever has any active stimuli for rules $R11a$ and $R11b$. In other words any execution of $\mathcal{R}$ starting from C^0 will contain no applications of either $R11a$ or $R11b$. Thus configuration C will have all the *1*-cells of its mixed row to the left of all the *0*-cells in that row (as was the case in C^0). This eliminates any stimuli for rules $R10a$ and $R10b$ from C.

■

Theorem 18 *The ruleset given in Fig. 3.10 is a stigmergic sorter for* TL*.*

Proof: We will show that the ruleset in Fig. 3.10 satisfies the two conditions in the definition of stigmergic sorter given in Sec. 2.5. Let C^0 be any configuration with a default auxiliary state matrix.

For condition 1, assume that C^0_V does not satisfy TL i.e. $\neg TL(C^0_V)$. By Lemma 12, the total cost of $C^0 \neq 0$. By Lemma 13, C^0 contains an active stimuli for some rule in $\mathcal{R}$. Thus C^0 is not a terminal configuration for $\mathcal{R}$. By Lemmas 14, 15 and 16, and a repeated application of Lemma 13, any execution of $\mathcal{R}$ that starts from C^0 will produce (at finite intervals) intermediate configurations with lower total costs than its predecessors. By Lemma 11 it follows that these executions must all terminate with configurations with total cost $= 0$. By Lemma 12, each of these terminal configurations satisfy TL.

For condition 2, assume that C^0 is a TL-configuration, i.e. $TL(C^0_V)$ holds. Using an argument similar to that for Condition 1, it is straightforward to show that every execution of $\mathcal{R}$ that starts from C^0 will eventually terminate. All we need show now is that these executions never affect the visible state matrix. By Lemma 17, it follows that none of these executions will ever contain any application of either of the five rules: $R10a, R10b, R11a, R11b$ and $R12$. But a careful observation of Fig. 3.10 shows that these five rules are the only rules of $\mathcal{R}$ that modify the visible state matrix. All the other rules can change only the auxiliary state matrix, leaving the visible state unaffected—thus satisfying condition 2.

■

Corollary 19 *TL is stigmergic-sortable.*

3.3 Fully Reachable pattern (FR)

This pattern specifies that every *1* value is reachable from any other *1* value in the configuration, i.e. there is a path made up entirely of *1*s between any two *1*s. See Fig. 3.13 for an illustration. To formally specify this pattern, we need to first define two relations between cells of a configuration as given below.

In an $m \times n$ configuration C, if $1 \leq i, i', i'' \leq m$ and $1 \leq j, j', j'' \leq n$ then

1. Cell (i, j) *is adjacent to* cell (i', j') if and only if $|i - i'| + |j - j'| = 1$, and

2. Cell (i, j) *is reachable from* cell (i', j') if and only if either $(i, j) = (i', j')$ or cell (i, j) is adjacent to a cell (i'', j'') such that $C_V[i, j] = C_V[i'', j'']$ and (i'', j'') is reachable from (i', j'). Clearly, this relation is symmetric, and any two cells reachable from each other have the same visible state.

Using the reachability relation defined above, we can now specify pattern FR. For an $m \times n$ configuration C, $FR(C_V)$ holds iff $\forall\ 1 \leq i, i' \leq m; 1 \leq j, j' \leq n :$

$$(C_V[i, j] \wedge C_V[i', j']) \implies (i, j) \text{ reachable from } (i', j').$$

Figure 3.13 shows examples of configurations in which FR does and does not hold.

Theorem 20 *FR is not locally checkable.*

Proof: The proof is exactly the same as the proof for TL (see Theorem 8), except that now C, C' and C'' are as shown in Fig. 3.14, and $\mathcal{S}_1 = \{d : (1, K_{col} + 1) \notin d\}$ and $\mathcal{S}_2 = \{d : (1, 1) \notin d\}$.

■

Corollary 21 *FR is not locally stigmergic-sortable.*

An outline of a non-local stigmergic sorter for FR is discussed in the next chapter.

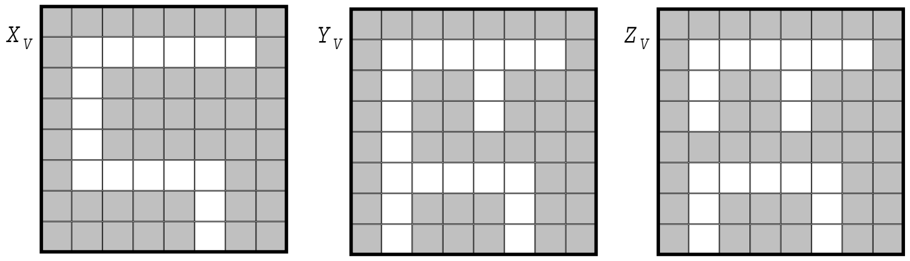


Figure 3.13 The visible state matrices of three 8×8 configurations called X, Y and Z. X and Y are FR-configurations, while Z is clearly not a FR-configuration—the *1* values in Z_V form two disconnected groups.

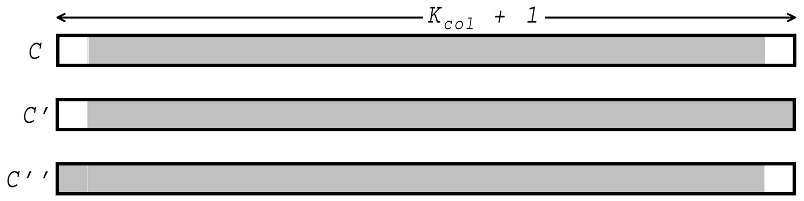


Figure 3.14 C, C' and C'' each are $1 \times (K_{col}+1)$ configurations. C is not a FR-configuration. C' and C'' each differ from C by the visible state of just one cell, and both are FR-configurations.

CHAPTER 4. Observations, conclusions and future work

The three pattern instances we covered in this work, can all be ordered by the relative strengths of their predicates. To formalize this ordering, consider a refinement relation between patterns, where P is a refinement of P' if P is a logically stronger predicate than P'. Thus a pattern P *refines* pattern P' if $P(C_V) \implies P'(C_V)$ for any configuration C. It can be easily verified that every TL-configuration is also a CG-configuration. With a little more effort it can be proven that every CG-configuration is also a FR-configuration. Thus we have the ordering: TL refines CG, and CG refines FR. From this ordering and Corollaries 7, 9 and 21 we can then make the observation that *local sortability of patterns is not monotonic with pattern refinement.*

Although the use of the cost function in the proof for the local sorter for CG was quite straightforward, in the case of TL, the argument for correctness of the non-local sorter using the cost function is quite involved. It seems that cost functions may be too weak for non-local stigmergic solutions, and it might be better to develop specialized proof techniques for such problems, perhaps by adapting more expressive distributed system formalisms like, input/out automata [11] into the stigmergic context.

We have a preliminary design for a relabeling ruleset that we think can stigmergically sort FR. The idea is to first (stigmergically) identify a distinguished *1*-cell—for example the one with the small row index followed by the smallest column index. Then using this particular cell as the root, the stigbots work outwards marking each unmarked adjacent *1*-cell with a special marking label. If for any marked *1*-cell, there are no more unmarked adjacent *1*-cells, then the labels reverse direction in that cell, flowing back to the root. Once all the outward going labels have flowed inwards to the root, all the *1*-cells reachable from the root have been marked, and the marking process is thus complete. At this point the root “broadcasts” another special label outward to “pull in” any unmarked (and thus unreachable) *1*-cells in the configuration. Since we do not yet have a formal proof of correctness of this scheme, we have the following conjecture.

Conjecture 22 *FR is stigmergic sortable.*

One of our future goals is to try and prove Conjecture 22, and use that experience in abstracting a specialized proof technique for stigmergic sorters. Another goal would be to prove that local checking is sufficient for local sortability in universal patterns. It also seems worthwhile to investigate non-universal patterns like, for example, “checkerboard” which is clearly locally checkable, but does not seem to be locally sortable (the proof for this can perhaps use the notion of indistinguishability). Thus it seems that in non-universal patterns local checkability may not be a sufficient condition. Finally, a more long term goal is to devise a comprehensive theory of the stigmergic computational paradigm.

BIBLIOGRAPHY

[1] Raghav Aras, Alain Dutech, and Francois Charpillet. Stigmergy in multi agent reinforcement learning. In *HIS '04: Proceedings of the Fourth International Conference on Hybrid Intelligent Systems*, pages 468–469, Washington, DC, USA, 2004. IEEE Computer Society.

[2] Eric Bonabeau, Sylvain Gurin, Dominique Snyers, Pascale Kuntz, and Guy Theraulaz. Three-dimensional architectures grown by simple 'stigmergic' agents. *Biosystems*, 56(1):13 – 32, 2000.

[3] Scott Camazine, Jean-Louis Deneubourg, Nigel R. Franks, James Sneyd, Guy Theraulaz, and Eric Bonabeau. *Self-Organization in Biological Systems : (Princeton studies in complexity)*. Princeton Studies in Complexity. Princeton University Press, 2003.

[4] Shlomi Dolev. *Self-stabilization*. MIT Press, Cambridge, MA, USA, 2000.

[5] N. R. Franks and A. B. Sendova-Franks. Brood sorting by ants: distributing the workload over the work-surface. *Behavioral Ecology and Sociobiology*, 30(2):109–123, March 1992.

[6] P. Grassé. La reconstruction du nid et les interactions inter-individuelles chez les bellicositermes natalenis et cubitermes sp. la théorie de la stigmergie: essai d'interprétation des termites constructeurs. *Insectes Sociaux*, 6:41–83, 1959.

[7] Owen Holland and Chris Melhuish. Stigmergy, self-organization, and sorting in collective robotics. *Artificial Life*, 5(2):173–202, 1999.

[8] István Karsai and Zsolt Pénzes. Optimality of cell arrangement and rules of thumb of cell initiation in Polistes dominulus: a modeling approach. *Behav. Ecol.*, 11(4):387–395, 2000.

[9] Peter Korošec and Jurij Šilc. A stigmergy-based algorithm for continuous optimization tested on real-life-like environment. In *EvoWorkshops '09: Proceedings of the EvoWorkshops 2009 on Applications of Evolutionary Computing*, pages 675–684, Berlin, Heidelberg, 2009. Springer-Verlag.

[10] N. Lynch. A hundred impossibility proofs for distributed computing. In *PODC '89: Proceedings of the eighth annual ACM Symposium on Principles of distributed computing*, pages 1–28, New York, NY, USA, 1989. ACM.

[11] Nancy A. Lynch and Mark R. Tuttle. An introduction to input/output automata. *CWI Quarterly*, 2:219–246, 1989.

[12] Christos H. Papadimitriou. The serializability of concurrent database updates. *Journal of the ACM*, 26:631–653, 1979.

[13] Ralf Schnabel, Marcus Bischoff, Arend Hintze, Anja-Kristina Schulz, Andreas Hejnol, Hans Meinhardt, and Harald Hutter. Global cell sorting in the c. elegans embryo defines a new mechanism for pattern formation. *Developmental Biology*, 294(2):418 – 431, 2006.

[14] Ana B. Sendova-Franks, Samuel R. Scholes, Nigel R. Franks, and Chris Melhuish. Brood sorting by ants: two phases and differential diffusion. *Animal Behaviour*, 68(5):1095 – 1106, 2004.

[15] Fitzgerald Steele, Jr. and Geb Thomas. Directed stigmergy-based control for multi-robot systems. In *HRI '07: Proceedings of the ACM/IEEE international conference on Human-robot interaction*, pages 223–230, New York, NY, USA, 2007. ACM.

[16] George Varghese. *Self-stabilization by local checking and correction.* PhD dissertation, Massachusetts Institute of Technology, 1993.

[17] Jennifer E. Walter, Jennifer L. Welch, and Nancy M. Amato. Distributed reconfiguration of metamorphic robot chains. *Distrib. Comput.*, 17(2):171–189, 2004.

[18] Justin Werfel. *Anthills built to order: automating construction with artificial swarms.* PhD dissertation, Massachusetts Institute of Technology, Cambridge, MA, USA, 2006.

[19] Daniel Yamins and Radhika Nagpal. Automated global-to-local programming in 1-d spatial multi-agent systems. In *AAMAS '08: Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems*, pages 615–622, Richland, SC, 2008. International Foundation for Autonomous Agents and Multiagent Systems.