One-Dimensional Discrete Pattern Formation

Coding Exercise, CS266 Fall 2006

November 13, 2006

One-dimensional discrete patterns are relatively simple to describe. However, constructing distributed algorithms by which simple agents can *self-organize* such patterns involves some interesting challenges whose solutions provide insight into more complex local-to-global problems. This coding exercise walks you through the construction of some basic 1-D pattern formation algorithms. It is divided into four problems, each of which addresses a fundamental issue in local-to-global programming. (There is also an extra credit problem, which could if extended properly by the basis of a good final project as well.)

This exercise will be posted on the CS266 website in the morning of Monday, 11/13/2006. Write-ups will be due in class on Thursday, 11/30/2006. Find a coding-buddy and work on the exercise in pairs: the thinking and coding should be shared, but everyone must produce their own write-up.

The coding should be done in the Matlab simulation environment that I have developed and distributed, and I'd like you to produce the write-ups using the Matlab publication feature. If you're absolutely bent on using another coding environment, that's OK. But please re-port your results back into Matlab for the write up – unless you want to use Mathematica and its native publication feature, which is also nice. (Should you choose to do this, please make sure to faithfully replicate the data structures!)

Problem 1: Equigrouping on Stateless $\overrightarrow{\mathbb{Z}}$

Your first mission is to code up, verify, and briefly analyze a local rule that solves the p-equigrouping task. The underlying model is exactly that presented in the first lecture: the one-dimensional infinite directed line $\overrightarrow{\mathbb{Z}}$ equipped with finitely-many stateless mobile agents. As we discussed in class, a configuration is *p-equigrouped* if every agent is in a contiguous set of exactly p pairwise-adjacent agents, with at least one space between each such group.

To start with, find a local rule f (with $r(f) > 1$) such that for all initial conditions X_0 that are $r(f)$-connected and contain a multiple of p agents, and all legal call sequences s, the limit $\lim_s f_s^n(X)$ exists and is p-equigrouped. We described a bare-bones outline of a solution to p-equigrouping in class. An acceptable answer to this problem would formalize the algorithm we described and code it up, and then provide a clear proof of correctness. You're also welcome to solve the problem your own way, as long as you provide an understandable proof that it functions as advertized.

Having coded up and formally verified your algorithm, characterize its asymptotic run-time performance. Design a set of numerical experiments (i.e. choosing large sets of initial conditions and call sequences in some judicious way) and empirically determine how the average-case run-time of the algorithm depends on the number of agents. Then, analytically prove what the worst-case run-time must be. Explain the discrepancy.

Problem 2: Generalized Repeat Patterns on Stateless $\overrightarrow{\mathbb{Z}}$

Now, generalize your result from the previous section. For any finite subconfiguration Q in the $\overrightarrow{\mathbb{Z}}$ model, we defined in class the *repeat pattern generated by* Q to be the set of configurations consisting of finitely many copies of Q placed along $\overrightarrow{\mathbb{Z}}$ at various spacings from each other. We saw that if we think of each Q as a finite binary string, than p-equigrouping problem is the special case in which $Q = 1^p0$. We then described the rudiments of a solution to this problem.

Here, your task is first to complete and code up the general pattern solution we saw in class, and provide a correctness proof. Repeat the run-time characterization program for a few patterns of your choice.

Problem 3: Stochastic Local Rules

In class we only discussed deterministic local rules, with the exception of the short comment prompted by Ece's question. A probabilistic rule is a function

$$f : B_r(a, X) \mapsto (p_{-1}, p_1, p_0 = 1 - p_1 - p_{-1}),$$

that is, a map from local configuration views to discrete distributions over the set of (Left, Right, Stay) actions.

Definition 1 A probabilistic local rule f is a solution to the task (P_i, P_f) if for all $X_0 \in P_i$ and all call sequences S in the chosen synchrony model, the sequence of probabilities

$$P_n = Pr(f_S^n(X) \text{ is fixed and in } P_f)$$

satisfy $\lim_{n\to\infty} P_n = 1$.[1]

It was clear from class that designing deterministic rules that solve specified problems requires some care (mostly to ensure that bad periodic cycles do not arise). It turns out that life is much easier if you allow probabilistic solutions. In this problem your task is to find and code a probabilistic solution to the p-equigrouping problem. Prove that your solution works using simple ideas from probability theory (I will discuss these in section). Once you've found the solution, repeat the run-time characterization and compare. Describe the trade-off.

[1] If you're a stickler for mathematical exactness you might wonder what the random variable is in this definition. It's clear that one can put a measure on the set of configurations and then describe distributions over that measure space. A distribution of actions f then acts on distribution of configurations to produce a new distribution of configurations. An initial configuration X_0 defines a distribution $\mathbf{X}_0$ of configurations with a single probability point mass of 1 at the X_0. As f acts on $\mathbf{X}_0$, the distribution widens out. This is just like in quantum mechanics between measurements.

Extra Credit: Extend your answer (both the algorithm and the proof of correctness) to the general repeat pattern problem.

Problem 4: The State/Radius Trade-Off

In class, we discussed the idea of the state/radius trade-off for local check schemes. That is, a local check scheme of radius r with a given number of states m can be transformed into an equivalent one having higher r with fewer states or lower r with more states. In this problem, you are asked to apply these ideas to the 1-D pattern formation problems, and extend them from static local checkability to actual dynamic algorithms.

As we defined in class, a check scheme θ with state set S and radius r *covers* a check scheme θ' with state set S' and radius r' if there is a map $\gamma : S^{2r(\theta)+1} \to S'$ such that whenever $\theta(X) = 1$ then $\theta'(\gamma(X)) = 1$. We saw that any locally check can be covered by check schemes with radius $r = 1$ and large S and one with high r and $|S| = 1$. This fact describes the *radius/state tradeoff.*

We also saw a more obviously applicable way to think about this. We discussed the *slot model* of state in class, in which each agent's internal state was represented as a set of slots, each of which can take on a state value. If there are to be approximately m total state values with c symbols, then $l = log_c(m)$ slots are needed. Abstractly, each agent's internal state can be described as a l-tuple of values between 1 and c, that is, as an element of $\{1, \ldots, c\}^l$. This new model allowed us to think of properties as desired configurations disorder/order pairs (P_i, P_f) in which configurations were patterns of values in the various slots.

Suppose (P_i, P_f) is a property in the l-slot model, and let $k \geq l$. As an analog for local check schemes we consider maps

$$\theta : ([c]^k)^{2r+1} \longrightarrow \{0, 1\}$$

where $[c] = \{1, \ldots, c\}$. Such a θ is a local check scheme for (P_i, P_f) in the k-slot model, if whenever $\theta(X) = 1$ then X, restricted to the first l slots, is in P_f or not in P_i. That is:

$$\bigwedge_{a \in X} (\theta(B_{r(\theta)}(a, X)) = 1) \;\Rightarrow\; X|_{(1,\ldots l)} \in P_f \cup (\mathcal{C} \setminus P_i).$$

We saw that another way to state the one direction of the radius/state tradeoff was that for any locally checkable l-slot property, with check radius r and any $r' < r$, by choosing k large enough you find a k-slot local check scheme with for the property with radius r'. We also saw a canonical way to construct the tradeoff.

Now, this was a *static* result. That is, it only regarded the ability to make radius/state tradeoffs in static checking of a property and not the construction of an actual algorithm. Your task here is to bridge this gap. That is: design an algorithm whose input is the generator of repeat pattern P and a desired r, and whose output is an algorithm with radius r with some larger amount of state that solves P. [Hint: composing your results from problem 2 and the canonical construction of the tradeoff for static checking gets you *most* of the way, but a little bit of extra state might be required ....]

You can either solve this probem in the oriiginal model, or built a simple extension of the simulation framework to include the slot-model and then solve the problem there.

Extra Credit: Pattern Formation on $\overline{\mathbb{Z}}(6)$

In class, we described a model of mobile agents on the undirected infinite discrete line, $\overline{\mathbb{Z}}$. $\overline{\mathbb{Z}}$ is like $\overrightarrow{\mathbb{Z}}$ but without an underlying orientation built in to the graph description of the space. Hence, local agents are unable to access a ready-made global direction and if they need one, have to emerge it for themselves.

As we saw in class, $\overline{\mathbb{Z}}$ has some non-trivial symmetries that $\overrightarrow{\mathbb{Z}}$ does not have. These symmetries rule out the possibility of perfectly forming most patterns on $\overline{\mathbb{Z}}$ in the totally synchronous update model. One way to understand this is to note that the pattern formation algorithms on $\overrightarrow{\mathbb{Z}}$ rely on a global polarity to coordinate the wave of correctness that sweeps through the system. In $\overline{\mathbb{Z}}$, the system gets stuck in situations in which symmetry cannot be broken, and no coherent wave can be started.

The asynchronous update model provides just enough symmetry-breaking potential to solve problem and, with high enough radius, there is a solution to this problem for stateless agents (See problem 6). However, it is significantly technically simpler to solve this problem with a little help from some internal state. In class, we discussed allowing each agent to have a number m of internal states (corresponding to a RAM of size $log_2(m)$). The model of $\overline{\mathbb{Z}}$ in which agents has m states was denoted $\overline{\mathbb{Z}}(m)$.

Your mission, should you choose to accept it, is to design and code and algorithm for pattern generation on $\overline{\mathbb{Z}}$ that uses no more than six internal states, i.e on $\overline{\mathbb{Z}}(6)$. The solution breaks down into two natural parts: solving polarity generation in a non-mobile environment and then coupling polarity algorithm to a mobile pattern generation algorithm.

Non-mobile Polarity Generation

Definition 2 A function $O : \{1, \ldots, m\}^{2k+1} \to \{\pm 1\}$ is a *k-place orientation function with m states* if $O(\vec{a}) = O(\vec{b})$ then $O(\vec{a}_\dagger) = O(\vec{b}_\dagger)$, where the $\vec{x}_\dagger$ denotes x read off in reverse order.

Let X be a configuration in the $\overline{\mathbb{Z}}(m)$ model. The configuration *possesses a k-visible global orientation* if there is a k-place orientation function O such that if you list the agents in X from one end to the other, $a_1, \ldots, a_n$ then $O(st(a_{i-k}), st(a_{i-k+1}), \ldots, a_i, \ldots, a_{i+k})$ is the same for all i.

An initial conditions X_0 in the $\overline{\mathbb{Z}}$ model is $(2, r)$-*connected* if for all agents x, $d(x, x^{+2}) \leq r$, where x^{+m} are either of the m-th closest agents to x on its right or left.[2] Find a local state-change rule F_{pol} such that under the completely asynchronous update model, F_{pol} drives any $(2, r)$-connected configuration to state in which there is a 1-visible global orientation. (You're allowed to use the $m = 6$ model, but have the orientation function itself use fewer than 6 states.)

Coupling Polarity to Motion

Now, take your polarity generation algorithm from the previous problem and combine it with the algorithm you created in problem 2, to produce a solution to pattern formation on $\overline{\mathbb{Z}}(6)$ in the asynchronous udpate model.

Formally: find an algorithm that has as input the generator of a stateless repeat pattern P, and outputs a local rule F_P such that for all $r(F)$-connected initial conditions X_0 in $\overline{\mathbb{Z}}(6)$, and all totally asynchronous call sequences s, $\lim_{n\to\infty} F_S^n(X_0)$ exists and satisfies the pattern P. (NB: the

[2] Recall that "left" vs "right" are not well-defined in $\overline{\mathbb{Z}}$.

final state of your algorithm must have all of the internal state cleared, i.e. all the agents in the same internal state. The six states used to generate and hold the polarity must die off before your algorithm comes to a fixed state.)

Standing Offer

This problem can easily be extended to a nice theoretical final project if you want. If you're interested, let me know. I make the following offer regarding this: a well-done version of this done as a final project would form the nucleus of a nice conference paper, and I'd be honored to co-author with you in the event.