The Brueckner Network: An Immobile Sorting Swarm

William A. Tozier
bill@williamtozier.com

Michael R. Chesher
Department of Industrial and Operations Engineering
1205 Beal Avenue
University of Michigan
Ann Arbor, MI 48109-2117
mchesher@umich.edu

Tejinderpal S. Devgan
tsdevgan@umich.edu

ABSTRACT

In many industrial applications, the dynamic control of queuing and routing presents difficult challenges. We describe a novel ant colony control system for a multiobjective sorting problem using an Emergent Sorting Network (ESN) designed by Sven Brueckner. Here, an immobile population of extremely simple agents reside at fixed vertices of a network, passing parts through the network, and as a result sorting a stream of colored parts. We explore effects of network size, and the effect of task difficulty (number of colors sorted) on timing and sorting performance. We demonstrate an unexpected regime shift in the swarm's collective behavior caused by network filling effects, and show evidence that this effect is due to the creation of *ad hoc* buffer regions: transient task specialties arising among the homogeneous agents.

Categories and Subject Descriptors

I.211 [**Artificial Intelligence**]: Distributed Artifical Intelligence—*multiagent systems, coherence and coordination*

General Terms

Design, Experimentation, Theory

Keywords

Emergent Sorting Network; swarm intelligence; queuing theory; emergent behavior.

1. INTRODUCTION

We explore and develop an ant-colony optimization (ACO) approach to sorting a continuous queue, which was first described by Sven Brueckner [2,3]. As with all ACO algorithms [ref], this one solves the problem by framing the search in terms of the dynamics of a system of simple autonomous agents following *local* rules, interacting with one another through their "environment" to produce as an *emergent behavior* the desired solution to the global problem.

2. Brueckner networks

The Brueckner ESN is composed of a set of *identical routing robots*, connected to one another by *channels*. *Parts* with different characteristics (color, here) arrive in unsorted order and are inserted into the network via an *input queue*, pass from channel to robot to channel in turn, and exit the system via an *output queue*.

2.1 Robots

Whenever they are *activated*, a robot executes the following sequence of rules, in fixed order [2]:

Rule 1. ***Try to repeat***: The robot checks to see if it can move a part of the same color it moved last time from any input channel to the same output channel it used last time. If there are multiple parts of the correct color available, one is chosen at random. If a part is moved, then the logic terminates; if no part can be moved, then we continue to Rule 2.

Rule 2. ***Decide whether to sleep*****:** The robot counts the proportion of its output channels occupied (blocked) by parts. This fraction (which will fall in [0,1]) represents the probability that the robot will *sleep* (shut down) for a fixed period. If the robot does go to sleep, the logic ends here; if it does not, we move to Rule 3.

Rule 3. ***Move a random part***: The robot chooses a part from any occupied input channel, and moves it to any available output channel. We can infer that space is available somewhere downstream, since the robot is not asleep (and the probability of sleeping when all downstream channels are full is 1). Parts and channels are chosen with uniform probability from all possible alternatives. If a part is moved successfully, the robot memorizes its color and the output channel into which it was placed; if no part is available to be moved, the robots memory is retained.

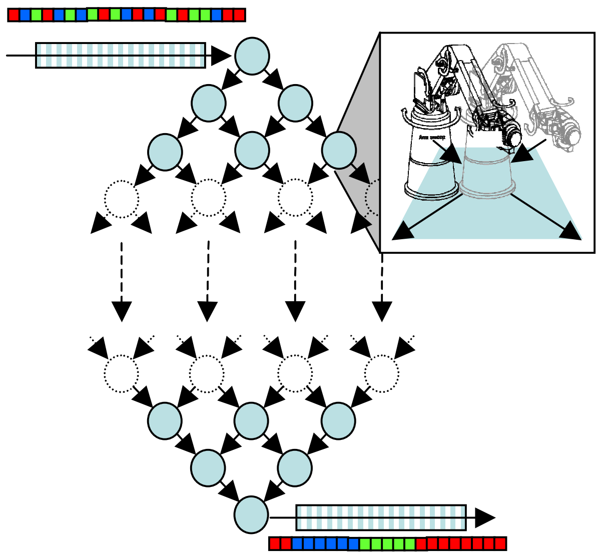

Figure 1. Emergent Sorting Network.

3. DATA COLLECTION

We collected and derived four measures of performance:

Output queue color changes: The total number of times a pair of adjacent parts in the output sequence had different colors.

Sortedness: Defined as $1-\frac{switches_o}{switches_i}$, where *switches*$_o$ is the number of color changes in the output queue, and *switches*$_i$ the number of color changes in the input queue.

Average (or max) length of runs: The average (or maximum) length of a continuous series of any one color.

Average time in system: the average time a part spends between the time it exits the input queue, and the time it enters the output queue.

4. RESULTS

4.1 Trade-offs between sortedness and delay

We collected performance data from networks size 2x2 to 9x9, sorting a series of 2000 parts of 6 colors, using update sequence 2. The results are shown in Figure 2. The number of color changes in the output queue of ~2000 parts (dark diamonds) drops, though it appears to be approaching a limit. The average number of minutes parts spend in the system rises roughly linearly with the number of robots.

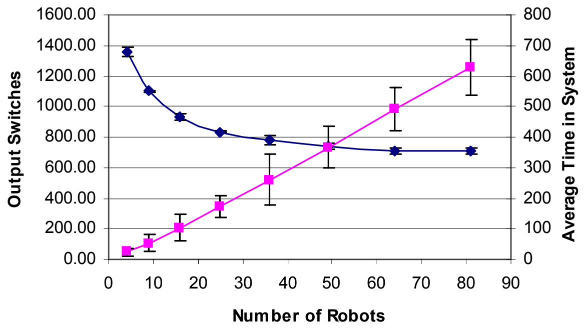

Figure 2: Tradeoff between sortedness and delay

4.2 Phase transitions & emergent buffering

This effect appears to be the one significant qualitative difference between discrete to continuous-time update schemes, and is we believe a *saturation* effect. When parts pass through the system individually—that is, when there are so few parts moving through the system that there are no delays, and no parts adjacent to each other—they are not sorted, *even though the memory of the robots is not ephemeral*. That is, a robot that last moved a blue part will remember the fact regardless of the time until it sees the next. The sorting effect occurs only when parts are waiting in the network.

5. DISCUSSION & FUTURE WORK

We suspect that the sorting behavior is due to the creation of these buffers in regions of the network close to the exit. This is consistent with the delay effects, as well: the lack of delay allows individual parts to fall through the system alone; no buffers are created, so no long chains of sorted parts can be constructed by the last router. At the other extreme, the jammed system is controlled by essentially random movements of the few active robots, moving isolated parts as they can. The constraint here is on available space to repeat actions, and the result is again that buffers cannot be maintained or managed.

The dynamic balance between filling, sorting, and delay is striking. The sudden transition between the relatively sparse regime, in which sorting is poor but retention time is low, and the crowded regime in which sorting is improved but delays are experienced, we believe is due to the ESN's use of *ad hoc storage buffers*.

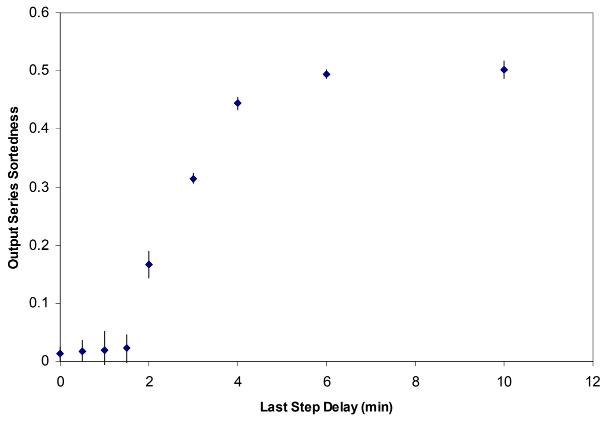

Figure 3: Saturation effect on sortedness

Thus, the effectiveness of the sorting network arises not from a specialization of certain paths in the network for certain colors, but rather the dynamic creation of pools of identical color in congruous channels, which can be drained in single runs. It seems clear that the principal effect of network topology on performance will occur through its effect on the structure of these buffers. We are pursuing this line of inquiry in current work.

6. REFERENCES

[1] Alexander, C. 1977. A Pattern Language : Towns, Buildings, Construction. Oxford University Press.

[2] Brueckner, S. 2000. Emergent Sorting. Software Demonstration at the Fourth International Conference on Autonomous Agents (Agents 2000), Barcelona, Spain. June, 2000. Paper, overview and software are available online at <http://www.erim.org/~sbrueckner/ publications.html>

[3] Brueckner, S. 2003. Engineering Complex Adaptive Systems for Real-World Applications., presented at the SIAM Conference on Mathematics for Industry, Challenges and Frontiers. October 23-25, Toronto, Canada.

[4] Huberman, B. A. and N. S. Glance. 1993. Evolutionary games and computer simulations. *Proc. Natl. Acad. Sci. USA* 90: 7716.

[5] McMullin, B. and F. Varela. 1997. Rediscovering Computational Autopoiesis. Presented at ECAL-97, Brighton, UK. Available online via <http://www.eeng.dcu.ie/~alife/bmcm-ecal97/bmcm-ecal97.html>

[6] Parunak, H. Van Dyke. 1997. "Go to the Ant": Engineering Principles from Natural Multi-Agent Systems. *Annals of Operations Research* 75: 69.