On the Seeding of Self-Reproducing Systems

Amor Menezes and Pierre Kabamba

Department of Aerospace Engineering
University of Michigan
Ann Arbor, Michigan 48109-2140

amenezes@umich.edu, kabamba@umich.edu

November 4, 2008

Abstract

This report is motivated by the need to minimize the payload mass required to establish an extraterrestrial robotic colony. One approach for this minimization is to deploy a colony consisting of individual robots capable of self-reproducing. An important consideration for the establishment of such a colony is the identification of a seed. Since self-reproduction is achieved by the actions of a robot on available resources, a seed for the colony consists of a set of robots and a set of resources. In this work, we discuss possible approaches to the seeding problem under certain assumptions, providing three novel algorithms to determine a seed for a self-reproducing system. Two of these algorithms find optimal seeds for particular classes of self-reproducing systems. Here, optimality is understood as the minimization of a cost function of the resources and robots. Illustrations of each algorithm are provided.

Contents

List of Figures

Chapter 1

Introduction

Recent scientific research in self-reproduction has raised the prospect of advances in such diverse areas as space colonization, bioengineering, evolutionary software and autonomous manufacturing. Inspired by the work of John von Neumann [1], extensive study of self-reproducing systems has taken place, including cellular automata, computer programs, kinematic machines, molecular machines, and robotic colonies. A comprehensive overview of the field is documented in [2] and [3].

Von Neumann postulated the existence of a threshold of complexity below which any attempt at self-reproduction was doomed to degeneracy. However, he did not define either complexity or degeneracy, nor did he compute the threshold's value. An extensive literature survey in [4] indicates that no one had published an evaluation of this threshold in the following 60 years. Recently, [5] developed a novel theory of generation that is able to compute this von Neumann threshold. The results included a necessary and sufficient condition for non-degenerate offspring, i.e., offspring with the same reproductive capability as the progenitor. Reference [6] presented a probabilistic version of these results, and also demonstrated parallels with information theory. References [7], [8], and [9] extended these results by providing algorithms that identified a seed for various classes of self-reproducing systems. Two of these algorithms produce optimal seeds. This work is a more detailed exposition on all three algorithms.

The remainder of this chapter presents a rationale for the identification of a seed for a self-reproducing system and discusses what makes the general seeding problem difficult. Chapters 2 through 4 highlight the results of solutions to seeding while examining their advantages and limitations; chapter 2 documents the Seed Identification and Generation Analysis (SIGA) algorithm, chapter 3 details the Restricted Seed Identification (RSI) algorithm, and chapter 4 documents the general Seed Identification (SI) algorithm. Each chapter presents the necessary assumptions and definitions for seed identification, analyzes the properties of a seeding algorithm, and provides at least one illustrative example. Chapter 5 presents conclusions.

1.1 Motivation

Within the context of extra-terrestrial colonization, current phased approaches to Martian exploration see the development of an enduring robotic presence on the Moon in the next five years. Several space agency roadmaps, of which [10] is typical, suggest that individual countries will deploy advanced robots on an as-needed basis to expand the size of an established colony. It is well known, however, that for every unit mass of payload to be launched into space, eighty additional units of mass must be launched as well [11] - hence the motivation to endow robots with the capacity for self-reproduction. These machines would be able to utilize on-site resources to enlarge their numbers when deemed necessary for a given task. Extra-terrestrial systems with such capability are less dependent than traditional colonies on the fiscal constraints of multiple launches of robots. Self-reproduction may therefore provide a highly cost-effective option for extra-terrestrial colonies.

To minimize mass, it would be even more efficient to identify the required initial elements for a self-reproducing system, and send the smallest quantity of these into space. The identification of this “seed” is the goal of this report.

1.2 Background

We first define the following terms for use throughout the work: reproduction, replication, self-reproduction, and self-replication. For a historical perspective of the first two terms, the reader is referred to Freitas' excellent discussion on the subject in [2]. We consider reproduction in biological systems to encompass the capacity for genetic mutations and evolution. Thus from an information standpoint, reproduction involves a change to the DNA code during the generation of progeny. Likewise, we will take *reproduction* in an artificial generation system to imply a change in the information specifications of an offspring. We reserve the term *replication* for progeny that have identical information content to that of the progenitor. *Self-reproducing* and *self-replicating* will be used to refer to those entities that perform the information equivalent of asexual reproduction or mitosis, i.e., the entities can reproduce or replicate based on the information specifications of only one progenitor.

The reader is referred to [5] for details of Generation Theory. Briefly, the theory formalizes self-reproduction by "machines," a term describing any entity that is capable of producing an offspring regardless of its physical nature. Hence a robot, a bacterium, or even a piece of software code is considered to be a machine in this theory if they can each produce another robot, bacterium or some lines of code respectively. These machines utilize resources to self-reproduce. A selected resource is manipulated by the parent machine via an embedded generation action to produce an outcome, which itself may or may not be a machine. Thus, we can state the following:

Definition 1.1. A *generation system* is a quadruple $\Gamma = (U, M, R, G)$, where

- U is a *universal set* that contains machines, resources and outcomes of attempts at self-reproduction;
- $M \subseteq U$ is a *set of machines*;
- $R \subseteq U$ is a *set of resources* that can be utilized for self-reproduction; and,

- $G : M \times R \rightarrow U$ is a *generation function* that maps a machine and a resource into an outcome in the universal set.

It is possible that $M \cap R \neq \oslash$, and also $M \cup R \neq U$, as illustrated in Fig. 1.1. The former implies that machines can belong to the set of resources, and the latter states that outcomes of attempts at generation may be neither machines nor resources.

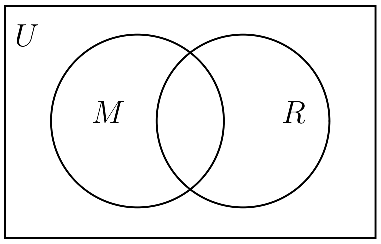

Figure 1.1: Pictorial representation of Definition 1.1.

When a machine $x \in M$ processes a resource $r \in R$ to generate an outcome $y \in U$, we write:

$$y = G(x, r). \tag{1.1}$$

In (1.1), we say that "x is capable of generating y," and we call the process *reproduction*. If we have $x = G(x, r)$ then we say that "x is capable of generating itself," and we call the process *replication*.

We also use concepts from graph theory [12] in this paper. Equation (1.1) may be represented by a *directed reproduction graph*, γ, as shown in Fig. 1.2. In this diagram, machine x and outcome y are vertices, resource r is an edge, and the direction of the edge indicates that machine x uses resource r to generate outcome y.

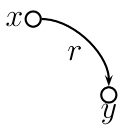

Figure 1.2: The directed reproduction graph of (1.1).

Definition 1.2. The *directed graph representation* of a generation system $\Gamma = (U, M, R, G)$ is the directed supergraph (V, E) containing all directed reproduction graphs that produce machines in M. Thus the vertex set, V, of the supergraph is the machine set, M, and the edge set, E, of the supergraph is the set $\{r \in R | y = G(x, r), x \in M, y \in M\}$.

Definition 1.3. The *generation sets* in a generation system are defined as:

- $M_0 = M$, the set of all machines;
- M_{i+1}, the set of all machines that are capable of producing a machine of M_i, $\forall i \geq 0$. That is, for $x \in M_{i+1}$, $\exists\, y \in M_i$ such that $y = G(x, r)$.

These sets are nested with the innermost generation set being important for self-reproduction. This set can be defined as:

$$M_\infty = \bigcap_{i=0}^{\infty} M_i. \tag{1.2}$$

It is shown in [5] that generation always proceeds outwards. Also, the notion of the rank of a generation system, as defined below, is emphasized.

Definition 1.4. The *rank of a generation system*, $\rho(\Gamma)$, where $\Gamma = (U, M, R, G)$ with generation sets M_i, $i \geq 0$, is the smallest integer ρ such that $M_\rho = M_{\rho+1}$. If $\forall i, M_i \neq M_{i+1}$, then the generation system has infinite rank.

For a generation system of finite rank ρ, the nesting of the generation sets stop at the integer ρ. All generation sets of order greater than ρ, including M_∞, are equal to M_ρ. A generation system that has a finite number of machines always has finite rank.

Definition 1.5. The *rank of a machine*, $\rho(x)$, in a generation system $\Gamma = (U, M, R, G)$ with generation sets M_i, $i \geq 0$, and $\rho(\Gamma) = \rho$, is equal to i if $x \in M_i \backslash M_{i+1}$ ("deficient generation rank"), or is equal to ρ if $x \in \bigcap_{i=0}^{\infty} M_i$ ("full generation rank").

Definition 1.6. A *generation cycle* is a sequence of generations resulting in the production of a machine identical to itself after n generations.

Machines capable of replication (i.e., a generation cycle of order one) in a generation system must belong to M_∞, and any exit from M_∞ is irreversible. It is possible for offspring machines to belong to M_∞ as long as their progenitors do as well. Thus the requirements for non-degenerate reproduction and replication are quantified. It is proved in [5] that there is a minimum threshold of rank above which a machine is able to generate an offspring without a decrease in generation rank. We call this the *von Neumann Rank Threshold*, τ_r, which satisfies

$$\tau_r = \rho(\Gamma). \tag{1.3}$$

The reader is referred to the material in [5] for proofs of the above statements, as well as many other insights into the information requirements of self-reproducing systems.

We can use a compact notation to denote a sequential selection of resources. Let (r_μ) be a sequence of μ resources from R. We define the notation

$$G\left(x, (r_\mu)\right) := G(\dots G(G(x, r_1), r_2) \dots, r_\mu).$$

1.3 Difficulty of the Seeding Problem

Many factors contribute to the inherent difficulty of seeding, including:

(a) the possibility that a given generation system is made up of multiple, disjoint subsystems, each with a different seed. Alternatively, there could be multiple, intersecting subsystems, with some common seed elements in each subsystem. Any seeding algorithm would have to be able to deal with both possibilities without any *a priori* knowledge about the system.

(b) the potential for generation cycles within a given self-reproducing system. If a cycle exists, then one wonders which machine in the cycle, if any, should belong to the seed.

(c) the fact that degenerate machines should not belong to the seed for a self-reproducing system. On the other hand, if all the machines in the generation system are degenerate, then one must identify a machine that is of highest generation rank to seed the system.

(d) the complexity of the resource set. A consistent theme in the literature is that a machine operates on an ordered list of elements constituting a resource. This list can include duplicates of elements contained in another resource that is also an ordered list. In addition, each list can also include machines.

(e) the existence of self-reproducing systems where the production of copies of a progenitor require the assistance of its offspring. Such systems are prevalent; examples include the Krebs cycle in a cell [13], the atmospheric ozone cycle and the atmospheric ozone cycle attacked by chlorine [14], the nitrogen cycle when starting a new aquarium [15], and some manufacturing systems as described in Appendix A. Typically, this phenomenon manifests itself as a combination of (b) and (d), when a resource employed at some stage of a generation cycle contains a machine that is generated at a different stage in the cycle. It is not always clear whether to include the progenitor, its offspring, or both, in the seed.

It is perhaps because of all of these factors that the seeding problem is still mostly open. We approach this problem by first considering a simplified generation system under restrictive assumptions, for which a seed is easily identified. This gives us the Seed Identification and Generation Analysis (SIGA) algorithm. The restrictive assumptions are then systematically relaxed to yield a more general seeding methodology.

Chapter 2

The SIGA Algorithm

To formulate the seed requirements of a self-reproducing system in a mathematically precise way, we make some assumptions about the generation system. These assumptions help structure the seeding problem but, in some cases, unfortunately make non-optimal seeds possible, as described.

2.1 Assumptions and Problem Definition

We start by listing the basic assumptions required to seed a generation system.

Assumption 2.1. Both the number of machines and the number of resources are finite.

Assumption 2.2. An inexhaustible supply of each resource is available.

Assumption 2.3. All the machines in the generation system must be produced, although they need not all belong to a seed.

We now make assumptions with respect to both machines and resources. We first assume that every resource in the set R of a generation system is utilized by a progenitor machine so that another machine can be produced.

Assumption 2.4. Let $\Gamma = (U, M, R, G)$ be a generation system. We assume that $\forall r \in R, \exists x \in M$ such that $G(x, r) \in M$.

This assumption simplifies the selection procedure of resources since it implies that all resources are necessary to produce offspring. Hence, a seeding algorithm can simply identify all possible resources as constituents of a seed. If we accept that all resources are necessary however, then we allow ourselves the possibility of selecting redundant resources. For instance, if there exist two resources such that a progenitor machine will produce the same offspring with each of those two resources, then by taking both resources in the seed, a redundant selection has been made and the resulting seed is non-optimal. We ignore this possibility and consider it an avenue for future refinement.

We allow for complexity within the resource set, and enable each resource to contain an ordered list of physical elements that may include machines. We therefore define a containment relation as follows.

Definition 2.1. If machine x_i belongs to an ordered list of the elements of resource r_j, then we say that x_i is *contained* in r_j, and we write $x_i \prec r_j$, where "$\prec$" is the *containment relation*.

Of course, if machine x_i is a resource itself, then this relation still holds true. We need the following definition as well, before stating an assumption on resources that has been made in all literature to date.

Definition 2.2. If machines $x_1, x_2, \ldots, x_\nu$ are contained in resource r, then we use the notation $r \backslash (x_1, x_2, \ldots, x_\nu)$ to refer to an ordered list of the elements of r that does not contain the machines $x_1, x_2, \ldots, x_\nu$.

Assumption 2.5. Let $\Gamma = (U, M, R, G)$ be a generation system. We assume that if machine x is contained in resource r, $x \prec r$, then the ordered list of the elements of r that does not contain the machine x also belongs to the set of resources, i.e., $r \backslash x \in R$.

Next, we assume that every machine in the generation system has a progenitor machine.

Definition 2.3. A *surjective generation system* is a generation system $\Gamma = (U, M, R, G)$ where $\forall y \in M, \exists x \in M$, and $\exists r \in R$ such that $y = G(x, r)$.

We further assume that there exists a machine in the generation system that is capable of producing any machine in the system after μ generations. This is a special case of a surjective generation system.

Assumption 2.6. We assume that in the generation system $\Gamma = (U, M, R, G)$, $\exists x_0 \in M$ such that $\forall x_1 \in M, \exists \mu_1 \leq \mu, \exists r_1, r_2, \ldots, r_{\mu_1}$ selected from R such that

$$G\left(x_0, (r_{\mu_1})\right) = x_1.$$

This rather restrictive assumption will give us the SIGA algorithm.

Definition 2.4. A *seed for* Γ *of order* μ is a set

$$\begin{aligned} S &= M_S \cup R_S, \text{ where} \\ M_S &= \{x_0\}, M_S \subseteq M, \text{ and} \\ R_S &= \{r_1, r_2, \ldots, r_\mu\}, R_S \subseteq R, \end{aligned}$$

such that $\forall y_1 \in M, \exists \mu_1 < \infty, \exists (r_{\mu_1}) \in R_S$, such that

$$G(x_0, (r_{\mu_1})) = y_1.$$

We can now state the following.

Seed Identification Problem 1. Given Γ, determine a seed under Assumptions 2.1 through 2.6.

2.2 Methodology

To solve Seed Identification Problem 1, we need to remove all degenerate machines from the sets M and R to determine M_S and R_S, and select one of the remaining non-degenerate machines to be x_0. The approach to developing a seed identification algorithm is similar to the Generation

Analysis Algorithm (GAA) [5], and in fact utilizes the GAA in its operation. The GAA employs the concept of an *outer layer*, first introduced in [5] and defined as follows.

Definition 2.5. The *outer layer* of a generation system $\Gamma = (U, M, R, G)$ is the set $M_0 \backslash M_1$. This is the set of machines such that, no matter what resource they use, they produce an offspring that is no longer a machine, i.e.,

$$M_0 \backslash M_1 = \{x \in M : \forall r \in R, G(x, r) \notin M\}.$$

After an outer layer is removed, a generation system of reduced rank remains. The GAA works by “peeling away” the outer layers of each of the resultant generation systems. We apply a similar notion to the development of the SIGA algorithm, having the algorithm peel away outer layers in both the set of machines and the set of resources, before picking one machine and the remaining resources to belong to the seed set.

By Assumption 2.4, any resource that does not contain a machine is assigned to be the seed. Next, the machines in the outer layer, $M_0 \backslash M_1$, are of lowest rank, do not help to perpetuate the system, and hence should not be in the seed, S. Thus, the outer layer of the machine set needs to be identified.

Let $M = \{x_1, x_2, \ldots, x_n\}$ and consider the *Descendancy Matrix*, defined as the $n \times n$ matrix of integers, D, such that

$$D_{ij} = \begin{cases} 1, & \text{if } \exists r \in R : x_j = G(x_i, r); \\ 0, & \text{otherwise,} \end{cases} \tag{2.1}$$

that is, $D_{ij} = 1$ if machine x_i is capable of generating machine x_j, and $D_{ij} = 0$ otherwise.

Now let $R = \{r_1, r_2, \ldots, r_m\}$ and consider the *Containment Matrix*, defined as the $n \times m$

matrix of integers, C, such that

$$C_{ij} = \begin{cases} 1, & \text{if } x_i \prec r_j; \\ 0, & \text{otherwise,} \end{cases} \tag{2.2}$$

that is, $C_{ij} = 1$ if machine x_i is contained in resource r_j, or is indeed a resource itself, and $C_{ij} = 0$ otherwise.

Let the *Seed Matrix*, be defined as the $n \times (n + m)$ matrix of integers, Σ, such that

$$\Sigma = \begin{bmatrix} D & C \end{bmatrix}. \tag{2.3}$$

Then the set of resources that do not contain any machine consists of those resources such that the corresponding columns of matrix C are zero. These columns may be removed from Σ, and the respective resources included in S. The outer layer of M consists of those machines in the matrix D that have corresponding rows of zeros. These rows, and the corresponding machine columns (even if not all zero), may be removed from Σ, and the respective machines added to $\bar{S}$, the set that is not the seed. If any of the removed machines exactly equals one of the resources, then that corresponding column may be removed from C as well.

We are now left with a reduced order generation system, that can be seeded in a similar fashion. The process of removing resources in the containment matrix, followed by removing lower-rank machines in the descendancy matrix and possibly in the containment matrix too, can be repeated to deflate the seed matrix until there are no more resources to remove. In each iteration, the columns of zeros in the matrix C denote those resources that are devoid of lower-rank machines. Of course, if a particular resource is nothing but a lower-rank machine, then this algorithm removes it from inclusion in the seed.

Once the iterations are over, one of two conditions may occur. It could be that all columns of the containment matrix have been removed, leaving nothing but the descendancy matrix. Thus all μ resource elements of S have been found, with $\mu \leq m$. If D can be further deflated, then this should be continued to obtain the machine of highest rank. When deflations of D are no longer possible, the machine of highest rank can be added as the x_0 required by S. If several machines are of equally high rank, then any one of these machines may be selected as x_0.

If, on the other hand, there are still resource columns left, but they cannot be removed due to the presence of a 1, then each resource can now be added to S as long as we deal with the corresponding machines that the resource requires (the rows with the 1). These machines are all of equal rank. By Definition 2.4, we can pick any one of these machines to be the required x_0 for S.

In short, of the difficulties listed in Section 1.3, (c) is effectively handled, (b) and (d) are ineffectively handled, and (a) and (e) are not handled.

The SIGA algorithm as stated is guaranteed to stop after a finite number of steps. This is because each iteration removes elements from a set with finite cardinality, stopping once the set is depleted. We summarize the algorithm in Section 2.3.

2.3 Pseudocode

Input: a generation system $\Gamma = (U, M, R, G)$, where the sets $M = \{x_1, x_2, \ldots, x_n\}$ and $R = \{r_1, r_2, \ldots, r_m\}$ satisfy Assumptions 2.1 through 2.6.

Outputs: the sets $(M_0 \backslash M_1)$, $(M_1 \backslash M_2)$, …, $(M_{\rho-1} \backslash M_\rho)$, $M_\rho = M_\infty$, the von Neumann rank threshold $\tau_r = \rho(\Gamma)$, the seed set S, and its order μ.

```
1: Compute the $n \times n$ descendancy matrix $D$, the $n \times m$ containment matrix $C$, and the $n \times (n+m)$ seed matrix $\Sigma$.
2: Initialize $i = 0, \mu = 0$.
3: **while** $R$ is not empty, and $C$ has at least one column of zeros, and $M$ is not empty **do**
4:     **for** each column of zeros in $C$ **do**
5:         Add $r_j$ to $S$.
6:         $\mu \leftarrow \mu + 1$.
7:     **end for**
8:     Update $R$ by removing the resource elements corresponding to zero columns of $C$.
```

```
 9:     Update $\Sigma$ by removing the corresponding zero columns of $C$.
10:     **return** $(M_i \backslash M_{i+1})$, the set of machines corresponding to zero rows of $D$.
11:     Update $M$ by removing the machines corresponding to zero rows of $D$.
12:     Update $\Sigma$ by removing the zero rows and corresponding zero columns of $D$, and any
        columns in $C$ for which the resource exactly equals the machine that has a zero row in
        $D$.
13:     **if** machines have been removed in this iteration **then**
14:         $i \leftarrow i + 1$.
15:     **end if**
16: **end while**
17: **if** $R$ is empty **then**
18:     **while** $M$ is not empty and $D$ has at least one row of zeros **do**
19:         **return** $(M_i \backslash M^{\epsilon}_{i+1})$, the set of machines corresponding to zero rows of $D$.
20:         Update $M$ by removing the machines corresponding to zero rows of $D$.
21:         Update $\Sigma$ by removing the zero rows and corresponding zero columns of $D$.
22:         **if** machines have been removed in this iteration **then**
23:             $i \leftarrow i + 1$.
24:         **end if**
25:     **end while**
26:     **return** $\mu$.
27:     **return** $(M_\infty) \leftarrow M$.
28:     **if** $|M_\infty| > 1$ **then**
29:         Pick an element of $(M_\infty)$ to add to $S$.
30:     **else**
31:         Add machine in $(M_\infty)$ to $S$.
32:     **end if**
33: **end if**
```

```
34: **if** $C$ does not have a column of zeros **then**
35:     **while** $M$ is not empty and $D$ has at least one row of zeros **do**
36:         **return** $(M_i \backslash M_{i+1})$, the set of machines corresponding to zero rows of $D$.
37:         Update $M$ by removing the machines corresponding to zero rows of $D$.
38:         Update $\Sigma$ by removing the zero rows and corresponding zero columns of $D$.
39:         **if** machines have been removed in this iteration **then**
40:             $i \leftarrow i + 1$.
41:         **end if**
42:         **for** each column of zeros in $C$ **do**
43:             Add $r_j$ to $S$.
44:         **end for**
45:         Update $R$ by removing the resource elements corresponding to zero columns of $C$.
46:         Update $\Sigma$ by removing the corresponding zero columns of $C$.
47:     **end while**
48:     **for** each remaining column in $C$ **do**
49:         Add $r_j \backslash (x \in M)$ to $S$.
50:         $\mu \leftarrow \mu + 1$.
51:         Pick a machine for which $r_j$ has a one in its row to $S$.
52:         Update $R$ by removing these resources.
53:         Update $\Sigma$ by removing the last columns of $C$.
54:     **end for**
55:     **return** $\mu$.
56:     **return** $(M_\infty) \leftarrow M$.
57: **end if**
58: **return** $\tau_r \leftarrow i$.
```

2.4 Example Application

We can use Generation Theory and the SIGA algorithm to analyze the *Semi-Autonomous Replicating System* designed by Chirikjian et al. [16, 17].

We take M to be the set of all entities that are made up of two or more LEGO Mindstorm kit components fixed together in some way. Let

$$\begin{aligned} M &= \{x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9\}, \text{ and} \\ R &= \{r_1, r_2, r_3, r_4, r_5, r_6, r_7, r_8, r_9\}, \end{aligned}$$

where we define each of the constituent machines and resources in the following manner. The sequence of generation steps is also outlined. The replication process is illustrated in Fig. 2.1 [16].

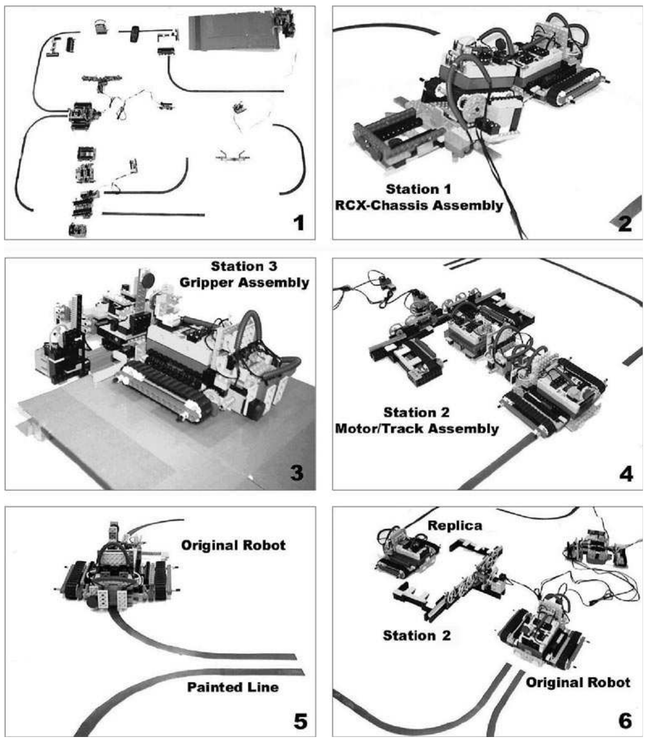

Figure 2.1: Replication process of the Suthakorn-Kwon-Chirikjian robot.

x_1 := prototype robot

r_1 := (conveyor-belt/sensor unit, docking unit, electrical connector, central controller unit

(CCU), electrical cable)

x_2 := chassis assembly station

$x_2 = G(x_1, r_1)$

r_2 := chassis

x_3 := chassis aligned in assembly position

$x_3 = G(x_1, r_2)$

r_3 := (robot control system, x_3)

x_4 := RCX-chassis assembly

$x_4 = G(x_2, r_3)$

r_4 := gripper assembly/disassembly station := (CCU, electrical connector, ramp and lift system, gripper)

x_5 := prototype robot with gripper

$x_5 = G(x_1, r_4)$

$x_1 = G(x_5, r_4)$

r_5 := (left LEGO hook, right LEGO hook, CCU, electrical connector, stationary docking sensor, motorized pulley unit)

x_6 := motor and track assembly station

$x_6 = G(x_5, r_5)$

r_6 := (left LEGO track, right LEGO track)

x_7 := tracks aligned onto hooks

$x_7 = G(x_1, r_6)$

r_7 := (motor/sensor unit, x_4)

x_8 := RCX-chassis-motor assembly, moved to position

$x_8 = G(x_1, r_7)$

r_8 := (x_7, x_8)

x_9 := prototype robot on hooks

$x_9 = G(x_6, r_8)$

$r_9 := x_9$

$x_1 = G(x_1, r_9)$

It follows that we have the generation diagram indicated in Fig. 2.2.

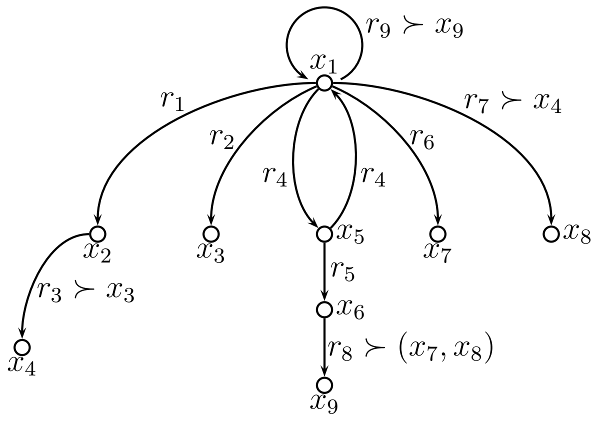

Figure 2.2: Directed graph representation of the Suthakorn-Kwon-Chirikjian robot.

With the SIGA algorithm,

$$D_0 = \left[\begin{array}{c|ccccccccc} & x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & x_8 & x_9 \\ \hline x_1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ x_2 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ x_3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ x_4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ x_5 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ x_6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ x_7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ x_8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ x_9 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right],$$

$$C_0 = \left[\begin{array}{c|ccccccccc} & r_1 & r_2 & r_3 & r_4 & r_5 & r_6 & r_7 & r_8 & r_9 \\ \hline x_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ x_2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ x_3 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ x_4 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ x_5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ x_6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ x_7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ x_8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ x_9 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right],$$

$$\Sigma_0 = \begin{bmatrix} D_0 & C_0 \end{bmatrix},$$

so that r_1, r_2, r_4, r_5 and r_6 are immediately identified as part of the seed. The descendancy matrix D_0 can also be deflated, yielding

$$M_0 \backslash M_1 = \{x_3, x_4, x_7, x_8, x_9\}.$$

We are left with

$$D_1 = \left[\begin{array}{c|cccc} & x_1 & x_2 & x_5 & x_6 \\ \hline x_1 & 1 & 1 & 1 & 0 \\ x_2 & 0 & 0 & 0 & 0 \\ x_5 & 0 & 0 & 0 & 1 \\ x_6 & 0 & 0 & 0 & 0 \end{array}\right],$$

$$C_1 = \left[\begin{array}{c|cccc} & r_3 & r_7 & r_8 & r_9 \\ \hline x_1 & 0 & 0 & 0 & 0 \\ x_2 & 0 & 0 & 0 & 0 \\ x_5 & 0 & 0 & 0 & 0 \\ x_6 & 0 & 0 & 0 & 0 \end{array}\right],$$

$$\Sigma_1 = \begin{bmatrix} D_1 & C_1 \end{bmatrix},$$

so that $r_3 \backslash x_3$, $r_7 \backslash x_4$, $r_8 \backslash (x_7, x_8)$, and $r_9 \backslash x_9$ now belong to the seed, and also

$$M_1 \backslash M_2 = \{x_2, x_6\}.$$

Since R is now empty, we continue to operate only on the machine set.

$$D_2 = \left[\begin{array}{c|cc} & x_1 & x_5 \\ \hline x_1 & 1 & 1 \\ x_5 & 0 & 0 \end{array}\right],$$

giving us

$$M_2 \backslash M_3 = \{x_5\}.$$

Lastly, the matrix that is left is

$$D_3 = \left[\begin{array}{c|c} & x_1 \\ \hline x_1 & 1 \end{array}\right],$$

which cannot be further reduced, yielding

$$M_3 = M_\infty = \{x_1\},$$

and $\tau_r = 3$. The seed set of order 9 for this system is

$$S = \{x_1\} \cup \{r_1, r_2, r_3\backslash x_3, r_4, r_5, r_6, r_7\backslash x_4, r_8\backslash(x_7, x_8), r_9\backslash x_9\}.$$

We have thus obtained a very logical, yet informative result - the original robot is (of course!) needed to initiate the system, assuming the existence of plentiful resources.

2.5 Limitations

The two immediate limitations in the SIGA algorithm stem directly from Assumption 2.6. There are many instances where one machine is incapable of producing all the other machines in the generation system; a simple example is the naturally occurring "biological cycle" required to start an aquarium [15]. The generation diagram of this self-reproducing system is illustrated in Fig. 2.3, using the model:

$$\begin{aligned} M &:= \{x_1, x_2, x_3, x_4\}, \text{ and} \\ R &:= \{r_1, r_2, r_3, r_4\}, \end{aligned}$$

where we define each of the constituent machines and resources in the following manner. The sequence of generation steps is also outlined.

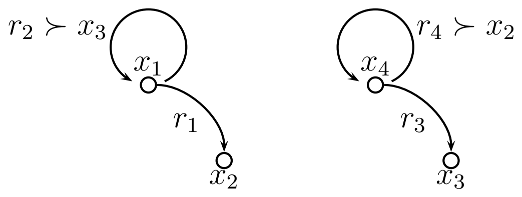

Figure 2.3: Directed graph representation of the biological cycle in an aquarium.

x_1 := algae

r_1 := (chlorophyll,sunlight,water)

x_2 := oxygen

$x_2 = G(x_1, r_1)$

x_3 := carbon dioxide

$r_2 := x_3$

$x_1 = G(x_1, r_2)$

x_4 := aerobic bacteria

r_3 := aquarium detritus

$x_3 = G(x_4, r_3)$

r_4 := (x_2,organic waste as ammonia or nitrite)

$x_4 = G(x_4, r_4)$

In the above example, it is clear that selecting only one of x_1 and x_4 will result in an incomplete machine seed set.

Secondly, suppose that there is more than one machine that is non-degenerate and capable of producing all the other machines in the generation system. Of these machines, the SIGA algorithm will pick one that is contained in the resources of R. If more than one of these machines is contained in the resources of R, then the algorithm arbitrarily picks a seed machine. If one considers a generation cycle of n machines and n resources where each resource contains any one of the n machines, then the SIGA algorithm breaks down because it is possible that the seed machine does not have a full resource for the first generation step.

These shortcomings are addressed in the Restricted Seed Identification (RSI) algorithm.

Chapter 3

The RSI Algorithm

Here, we take a more general approach to the seeding problem, and present necessary and sufficient conditions to find an optimal seed for a larger class of generation systems.

3.1 Assumptions and Problem Definition

Again, we start by listing the basic assumptions required to seed a generation system. Some of the assumptions and definitions in this chapter are a repetition, but are included for chapter completeness.

Assumption 3.1. Both the number of machines and the number of resources are finite.

Assumption 3.2. An inexhaustible supply of each resource is available.

Assumption 3.3. All the machines in the generation system must be produced, although they need not all belong to a seed.

As in the previous chapter, we can make the following assumption about the containment of machines in resources.

Assumption 3.4. Let $\Gamma = (U, M, R, G)$ be a generation system. We assume that if machine x is contained in resource r, i.e., $x \prec r$, then the ordered list of the elements of r that does not contain the machine x also belongs to the set of resources, i.e., $r \backslash x \in R$.

The given self-reproducing system may not be surjective, and even if it is, we may not be able to use the SIGA algorithm because Assumption 2.6 may fail. To formulate seeding requirements in a mathematically precise way, we begin by defining a seed, selecting the class of generation systems that we will deal with, and then using the properties of these systems to help setup the seeding problem.

Definition 3.1. Let $\Gamma = (U, M, R, G)$ be a generation system. A *seed of order* $\nu\mu$ for Γ is a set

$$\begin{aligned} S &= M_S \cup R_S, \text{ where} \\ M_S &= \{x_1, x_2, \ldots, x_\nu\}, M_S \subseteq M, \text{ and} \\ R_S &= \{r_1, r_2, \ldots, r_\mu\}, R_S \subseteq R, \end{aligned}$$

such that $\forall y_1 \in M, \exists \mu_1 < \infty, \exists (r_{\mu_1}) \in R_S, \exists y_0 \in M_S$, such that $G(y_0, (r_{\mu_1})) = y_1$.

In the rest of this chapter, we design an algorithm to produce a seed as per the above definition. The idea is to reduce certain generation systems into a form that can be easily seeded. We need a few more definitions before we can indicate the type of generation systems our algorithm is restricted to.

Definition 3.2. The generation system $\Gamma = (U, M, R, G)$ is *strongly regular* if whenever $y = G(x, (r_{\mu_1}))$, where x and y are machines and (r_{μ_1}) is a sequence of μ_1 resources, we have $y \not\prec r$ for all $r \in (r_{\mu_1})$.

Thus, in a strongly regular generation system, no machine can be contained in its ancestry.

Definition 3.3. A *family* is a generation system $\Gamma = (U, M, R, G)$ where for all $(x, y) \in M$, there exists $z \in M$, and $(r_n), (r_m) \in R$ such that $x = G(z, (r_n))$ and $y = G(z, (r_m))$. A *matriarch* of a family is an element $x_0 \in M$ such that for all $x_1 \in M, x_1 \neq x_0$, there exists (r_{μ_1}) selected from R such that $G(x_1, (r_{\mu_1})) = x_0$.

Note that empty sequences are allowed in the above definition of a family, so that it is possible to consider either x or y to be the common ancestor z.

Theorem 3.1. *Every family has a matriarch.*

Proof. See Appendix B. □

Theorem 3.2. *The directed graph representation of a family is weakly connected.*

Proof. See Appendix B. □

The converse to Proposition 3.2 is not true: a generation system whose digraph is weakly connected need not be a family (see Fig. 3.1 for a counter-example).

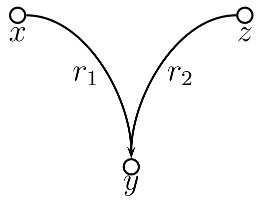

Figure 3.1: A weakly connected digraph representing two families.

Assumption 3.5. We assume that the generation system to be seeded, $\Gamma = (U, M, R, G)$, is strongly regular and made up of one or more disjoint families.

We can now state the following.

Seed Identification Problem 2. Given Γ, minimize the total cost of the machines and resources in a seed under Assumptions 3.1 through 3.5.

3.2 Methodology

Assumption 3.5 helps setup the seeding problem when we note that the underlying undirected graph of a family is connected. Since the connected components of a graph are the equivalence classes of the path existence relation between two vertices [12], and the directed graph representation of the generation system to be seeded is made up of one or more connected components, we can partition this graph into its connected components. Thus, for the class of systems in Assumption 3.5, seeding the whole generation system may be accomplished by seeding each individual family.

To seed by family, we need to determine all the descendants of a particular machine. This is facilitated by the notion of a "generation subsystem of a machine," which is a subset of a particular family and is itself a family.

Definition 3.4. The *generation subsystem of machine* x_0 is the generation system

$$\Gamma_{x_0} = (U, M_{x_0}, R_{x_0}, G)$$

where,

$$\begin{aligned}
M_{x_0} &= \bigcup_{i=0}^{\infty} M_{x_0}^i \\
M_{x_0}^i &= \{x \in M : \exists (r_i) \in R : x = G(x_0, (r_i))\} \\
R_{x_0} &= \bigcup_{i=0}^{\infty} R_{x_0}^i \\
R_{x_0}^i &= \bigcup \{(r_i) \in R : G(x_0, (r_i)) \in M\}.
\end{aligned}$$

In Definition 3.4, $M_{x_0}^i$ is the set of all the descendants of x_0 produced after i generations, M_{x_0} is the set of all the descendants of x_0, $R_{x_0}^i$ is the set of all resource sequences of length i that would produce a descendant of x_0, and R_{x_0} is the set of all resource sequences that would produce a descendant of x_0. Hence, the generation subsystem of x_0 is the largest family for which x_0 is a matriarch.

The idea for the RSI algorithm is to determine the subsystems for which there exists one machine capable of generating all other machines in the subsystem. It is among these subsystems that one may find a matriarch of a family. Consequently, individually seeding each of these subsystems of matriarchs seeds the whole family. We will let $M_{\female}$ denote the set of matriarchs.

In the generation system of a matriarch, x_0, every machine in the subsystem can be produced except possibly x_0 itself. Thus, in the course of seeding the subsystem of x_0, the machine to pick for the seed set of the subsystem, S_{x_0}, is x_0. The rationale for this process of identifying one machine of highest rank that can generate every machine in its subsystem comes from the next two propositions.

A necessary condition to minimize $|M_S|$ is the following.

Theorem 3.3. *Let $\Gamma = (U, M, R, G)$ be a family, and S be a seed set of Γ for which $|M_S|$ is a minimum. Then for all $x \in M_S$,*

$$\rho(x) = \begin{cases} \rho, & \text{if } |M_\infty| > 0; \\ \max_{y \in M} \rho(y), & \text{if } |M_\infty| = 0. \end{cases}$$

Proof. See Appendix B. □

Corollary 3.1. *If $\Gamma = (U, M, R, G)$ is a family, and $|M_\infty| > 0$, then $|M_S| \leq |M_\infty|$. On the other hand, if $|M_\infty| = 0$, then an optimal M_S has $|M_S| = 1$.*

The first statement above follows because an optimal M_S can only include machines from M_∞. The second statement above follows because there is only one machine for which the maximum rank condition is satisfied, a consequence of the fact that generation always proceeds outwards [5].

A sufficient condition to minimize $|M_S|$ is the following.

Theorem 3.4. *Assume that the generation subsystem of machine x, $\Gamma_x = (U, M_x, R_x, G)$ is strongly regular. Then a seed set for Γ_x, S_x, where $M_{S_x} = \{x\}$ and $R_{S_x} = R_x \backslash M_x$ has the minimum $|M_{S_x}|$.*

Proof. See Appendix B. □

In a strongly regular family, if a machine is contained in a resource, then that resource cannot be utilized in any sequence of resources used to generate the machine. Hence, the notion of containment has no effect on the seeding process for these systems. This is why we restrict the class of generation systems in this chapter to those that are solely made up of strongly regular families.

Assuming the given self-reproducing system of n machines and m resources is strongly regular and made up of one or more disjoint families, the first step of the algorithm would be to find the generation subsystems of all the machines in M, i.e.,

Step 1

```
for all $x_i \in M$, $1 \leq i \leq n$ do
    Determine $\Gamma_{x_i}$.
end for
```

Since we can represent a generation system as a weighted, directed graph, we can use established concepts of graph theory in the subsystem identification process. With each machine (vertex) as a starting point (root) in the initial generation system (directed graph), we need to find the subsystem (maximally connected subgraph) that can be generated (reached from the root).

Two well known algorithms to compute the reachable components in a graph are the Breadth-First Search (BFS) and the Depth-First Search (DFS) algorithms [18–21]. Applying either of these algorithms to the graph of the generation system in Step 1 yields Γ_{x_i} for all $1 \leq i \leq n$. In these subsystems, there is one machine capable of generating all machines in M_{x_i} except possibly x_i itself. We now want to partition the initial generation system in the following manner.

Step 2

```
Select the $\Gamma_{x_i}$ where $|M_{x_i}| \geq |M_{x_j}|, \forall 1 \leq j \leq n$.
```

This Γ_{x_i} is the largest generation subsystem of the initial self-reproducing system. We consider this to be our primary generation subsystem, and regard the system where the machine set is $M \backslash M_{x_i}$ to be a secondary generation subsystem. The secondary subsystem requires the removal of all $x \in M_{x_i}$ from M. The idea is to seed our primary subsystem first, and then go back to the secondary subsystem and partition and seed iteratively. By Assumption 3.5, if M is strongly regular and made up of one or more disjoint families, then $M \backslash M_{x_i}$ is also strongly regular and made up of one or more disjoint families. That is, the removal of one of the connected components of the graph does not affect the remaining connected components of the graph since each connected component is disjoint from each other.

In Step 2, if there are two subsystems Γ_{x_i} and Γ_{x_j} with the same machine set, then both x_i and x_j are matriarchs for the same family. To ensure the optimal seeding of this family, we will need to compare the cost of the seed resources when the subsystem of x_i is a primary subsystem and when the subsystem of x_j is a primary subsystem.

However, before we can tackle seeding of a primary subsystem (and by extension, the seeding of all other partitions), we need to ensure that the subsystem under consideration has the property that each offspring is generated from only one resource. Thereafter, if we select all resources to be a part of the seed set for the subsystem, we have avoided any unnecessary selection of redundant resources.

Let $J : R \to \mathbb{R}$ be a cost functional representing the mass of a resource, or the quantity required of a resource, or the resource's availability, etc. We will again make use of a result in graph theory for the next step. In graph theory, a subgraph of a finite directed graph is called a *branching* if it has the following properties: it contains all the vertices of the original graph (spanning); it is circuit-free; and the number of edges entering any vertex is less than or equal to one. If the number of entering edges is zero for only one of the subgraph's vertices, r, and the remaining vertices all have only one edge entering them, then the branching is a directed tree with root r [20]. The problem of finding a branching for which the sum of the edge costs is optimal (a maximum) was solved independently in [22–24], is well-treated in [18, 20, 25], and can be efficiently implemented using [26].

We have a similar situation if we first add a new machine, x_i', so that all instances of the resources (edges) that are used to produce machine x_i (enter the root vertex) in the primary subsystem (directed graph) are now used to produce machine x_i'. We can then use the Chu-Liu-Edmonds algorithm to obtain a generation subsystem where Proposition 3.4 is still applicable, but where redundant resources are also not present.

The implementation of the optimal branching algorithm that is assumed requires that: 1) the directed spanning tree that is found has a minimum rather than a maximum cost, and 2) a root vertex is accepted as additional input, so that search for the tree starts from this root instead of the first entry in a vertex-edge incidence list. We utilize these notions in formulating Step 3 of the RSI algorithm.

Step 3

```
for all the matriarchs of the largest generation subsystem do
    if in the graph representation of $\Gamma_{x_i}$, $x_i$ has entering edges then
        Add a new vertex $x_i'$.
        Change these edges so that they now enter $x_i'$.
    end if
    Find the directed minimum spanning tree (DMST) in the graph of $\Gamma_{x_i}$ with root at $x_i$.
    $\Gamma_{x_imin} \leftarrow$ the DMST of $\Gamma_{x_i}$.
end for
Select the $\Gamma_{x_imin}$ for which $\sum_{r \in R_{x_imin}} J(r)$ is a minimum.
```

We can now seed the resultant self-reproducing subsystem.

Step 4

$S_{x_i} = \{x_i\} \cup (R_{x_imin} \backslash M_{x_i})$.

Next, we obtain the generation system that remains to be seeded.

Step 5

Remove all $x \in M_{x_i}$ from M.

We continue the process so far on the subsidiary generation subsystems, iterating from Step 2 until there are no more machines left in M. The entire seed set is the union of all the seed sets for the various generation subsystems.

Step 6

```
if $M \neq \oslash$ then
    Go to Step 2.
else
    $S \leftarrow \bigcup S_{x_i}$.
    Stop.
  end if
```

3.3 Properties

In this section, we make some claims about the Restricted Seed Identification algorithm and the resultant seed that is output. The proofs of these claims can be found in Appendix B.

Theorem 3.5. *The RSI algorithm is correct. That is, the output of the algorithm is guaranteed to be a seed for the given generation system.*

Proof. See Appendix B. □

Theorem 3.6. *The RSI algorithm is complete. That is, the algorithm is guaranteed to output a seed if one exists for the given generation system.*

Proof. See Appendix B. □

Theorem 3.7. *The RSI algorithm is guaranteed to stop after a finite number of iterations. The best-case time complexity for the operation of this algorithm is $O(3n + nm + 2m + 1)$ iterations. The worst-case time complexity for the operation of this algorithm is the worst of $O(n^2m + 4n + 2m)$ and $O(n^2 + 5n + m)$ iterations.*

Proof. See Appendix B. □

Theorem 3.8. *The RSI algorithm produces a seed that is optimal with respect to the number of machines and the cost of the resources in the seed.*

The assumption of disjoint families is required to ensure optimality of the seed resource set. Although the proposed algorithm also works for families that are not disjoint, no claims about optimality can be made. However, we conjecture that the resultant seed is close to optimal in that case. If the families in the system are not disjoint, then Step 6 of the RSI algorithm would need to iterate from Step 1 instead of Step 2.

Proof. See Appendix B. □

Theorem 3.9. *Given a family* $\Gamma = (U, M, R, G)$*, the size of the seed either increases or stays constant with expanding* M *or* R*.*

Proof. See Appendix B. □

Of the difficulties listed in Section 1.3, (b) and (c) are effectively handled with the RSI algorithm, (a) and (d) are ineffectively handled, and (e) is not handled.

3.4 Example Application

Let us use the RSI algorithm to analyze a modified version of the *Semi-Autonomous Replicating System*. The original Suthakorn-Kwon-Chirikjian generation system in Section 2.4 is a single family, and so the application of the RSI algorithm terminates after one iteration. If we take into account the necessity of batteries for operation, the application becomes non-trivial. We stipulate that the robot controller (RCX) runs on charged batteries, and that there is a battery charger running on a supply of readily available electricity. Thus, the modified system is:

$$\begin{aligned} M &= \{x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}\}, \\ R &= \{r_1, r_2, r_3, r_4, r_5, r_6, r_7, r_8, r_9, r_{10}\}, \text{ where} \end{aligned}$$

x_{10} := battery charger

r_{10} := (electricity, uncharged batteries)

x_{11} := charged batteries

$x_{11} = G(x_{10}, r_{10})$

and the definition for r_3 becomes

r_3 := (robot control system, x_3, x_{11})

It follows that we have the generation diagram indicated in Fig. 3.2.

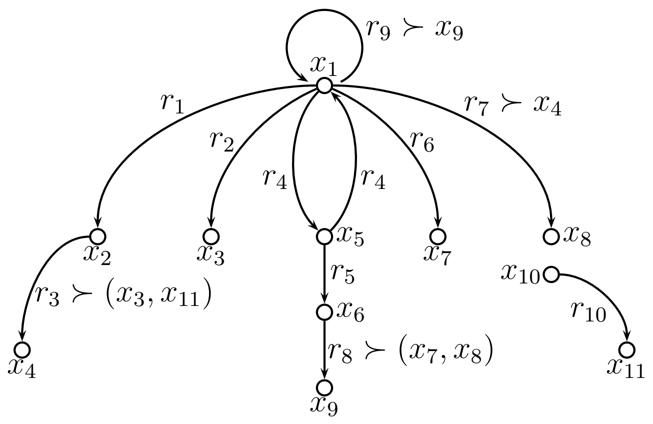

Figure 3.2: Directed graph representation of the modified Suthakorn-Kwon-Chirikjian robot.

This generation system is strongly regular, and is made up of two disjoint families. Applying the RSI algorithm to this generation system yields an optimal seed for the system. To demonstrate the workings of the algorithm, we give a part of the output at each step.

Step 1: The machine sets of $\Gamma_{x_i}, 1 \le i \le 11$ are the following.

$M_{x_1} = \{x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9\}$

$M_{x_2} = \{x_4\}$

$M_{x_3} = \emptyset$

$M_{x_4} = \emptyset$

$M_{x_5} = \{x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9\}$

$M_{x_6} = \{x_9\}$

$M_{x_7} = \emptyset$

$M_{x_8} = \emptyset$

$M_{x_9} = \emptyset$

$M_{x_{10}} = \{x_{11}\}$

$M_{x_{11}} = \emptyset$

Step 2: We can select either x_1 or x_5. Since the sets of machines that can be generated are equal, x_1 and x_5 must be matriarchs for the same family.

Step 3: For x_1,

$R_{x_1min} = \{r_1, r_2, r_3, r_4, r_5, r_6, r_7, r_8, r_9\}$.

The only choice made by the DMST algorithm is the selection of r_9 over r_4 in generating x_1', since the former has lower cost.

For x_5, $R_{x_5min} = \{r_1, r_2, r_3, r_4, r_5, r_6, r_7, r_8\}$.

The DMST algorithm does not select r_9.

Γ_{x_5min} is selected.

Step 4: For the first family, we get

$$\begin{aligned} S_{x_5} = \{x_5\} \quad \cup \quad & \{r_1, r_2, r_3 \backslash (x_3, x_{11}), r_4, r_5, r_6, \\ & r_7 \backslash x_4, r_8 \backslash (x_7, x_8)\}. \end{aligned}$$

Step 5: We are left with $M = \{x_{10}, x_{11}\}$.

Step 6: We now go back to Step 2.

Step 2: We select x_{10}.

Step 3: $R_{x_{10}min} = \{r_{10}\}$.

Step 4: For the second family we get

$$S_{x_{10}} = \{x_{10}\} \cup \{r_{10}\}.$$

Step 5: We are left with $M = \oslash$.

Step 6: A seed for this system is

$$\begin{aligned} S = \{x_5, x_{10}\} \quad \cup \quad & \{r_1, r_2, r_3 \backslash (x_3, x_{11}), r_4, r_5, r_6, \\ & r_7 \backslash x_4, r_8 \backslash (x_7, x_8), r_{10}\}. \end{aligned}$$

Thus, the battery charger and the prototype robot with gripper can initiate the semi-autonomous replicating system. Contrary to intuition, the optimal seed does not include the prototype robot together with resources, but instead includes the prototype robot with gripper and fewer required resources.

3.5 Limitations

Limitations of the RSI algorithm may be attributed to the requirement for strong regularity in the generation system. The computational complexity for checking whether a generation system is strongly regular is exponential in the number of machines and the rank of the system, and polynomial in the number of resources. Hence, depending on the size of the self-reproducing system, it may not be feasible to check for strong regularity.

As stated in the second flaw of the SIGA algorithm, there are instances where strong regularity is not satisfied; a simple example is the naturally occurring ozone cycle when it is attacked by chlorine in the atmosphere [14]. The generation diagram of this self-reproducing system is illustrated in Fig. 3.3, using the model:

$$\begin{aligned} M &:= \{x_1, x_2, x_3, x_4, x_5\}, \text{ and} \\ R &:= \{r_1, r_2, r_3, r_4, r_5\}, \end{aligned}$$

where we define each of the constituent machines and resources in the manner that follows. The sequence of generation steps is also outlined.

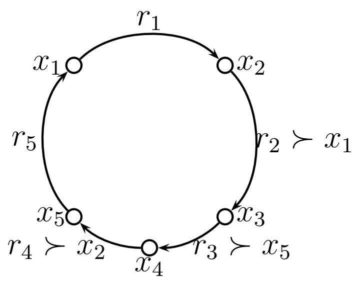

Figure 3.3: Directed graph representation of the ozone cycle being attacked by chlorine.

$x_1 := O_2$, or oxygen molecules

$r_1 :=$ ultraviolet radiation

$x_2 := O$, or excited oxygen atoms

$x_2 = G(x_1, r_1)$

$r_2 :=$ (x_1,neutral particle)

$x_3 := O_3$, or ozone molecules

$x_3 = G(x_2, r_2)$

$x_4 := ClO + O_2$

$x_5 := Cl + O_2$

$r_3 := x_5$ (note that just Cl is required)

$x_4 = G(x_3, r_3)$

$r_4 := x_2$

$x_5 = G(x_4, r_4)$

$r_5 := ()$, a dummy resource

$x_1 = G(x_5, r_5)$

In the above example, it is clear that a more powerful algorithm is desirable if such a self-reproducing system is to be seeded.

Chapter 4

The SI Algorithm

This chapter formulates the general seed identification problem and presents an extended version of the RSI algorithm, the SI algorithm.

4.1 Assumptions and Problem Definition

Just as in previous chapters, we start by providing a complete list of the assumptions required for seeding a generation system.

Assumption 4.1. Both the number of machines and the number of resources are finite.

Assumption 4.2. An inexhaustible supply of each resource is available.

Assumption 4.3. All the machines in the generation system must be produced, although they need not all belong to a seed.

Assumption 4.4. If machine x is contained in resource r, i.e., $x \prec r$, then the ordered list of the elements of r that does not contain the machine x also belongs to the set of resources, i.e., $r \backslash x \in R$.

We define a seed as in Section 3.1.

Definition 4.1. Let $\Gamma = (U, M, R, G)$ be a generation system. A *seed of order* $\nu\mu$ for Γ is a set

$$\begin{aligned} S &= M_S \cup R_S, \text{ where} \\ M_S &= \{x_1, x_2, \ldots, x_\nu\}, M_S \subseteq M, \text{ and} \\ R_S &= \{r_1, r_2, \ldots, r_\mu\}, R_S \subseteq R, \end{aligned}$$

such that $\forall y_1 \in M, \exists \mu_1 < \infty, \exists (r_{\mu_1}) \in R_S, \exists y_0 \in M_S$, such that $G(y_0, (r_{\mu_1})) = y_1$.

We now relax the notion of strong regularity.

Definition 4.2. The generation system $\Gamma = (U, M, R, G)$ is *weakly regular* if whenever $y = G(x, r)$, where x and y are machines and r is a resource, we have $y \not\prec r$.

Thus, in a weakly regular generation system, no machine can be contained in any resource used to produce that machine. In the next assumption, we will continue using the notions of families and matriarchs as introduced in Section 3.1.

Assumption 4.5. We assume that the generation system to be seeded, $\Gamma = (U, M, R, G)$, is weakly regular and made up of one or more disjoint families.

We can now state the following.

Seed Identification Problem 3. Given Γ, minimize the total cost of the machines and resources in a seed under Assumptions 4.1 through 4.5.

4.2 Methodology

The idea of the SI algorithm is that under Assumption 4.5, seeding the whole generation system may be accomplished by seeding each individual family, as before. To seed by family, we still need to determine the generation subsystems of each machine. However, the key difference is that in the course of seeding the subsystem of matriarch x_1, the machines to pick for the seed set of the subsystem, S_{x_1}, are x_1 and certain machines contained in R_{x_1}. The additional machines from R_{x_1} are required because we no longer have strong regularity.

The next theorem suggests an approach to seeding weakly regular generation systems, and replaces the sufficient condition for minimizing $|M_S|$ that was used by the RSI algorithm.

Theorem 4.1. *Assume that the generation subsystem of machine x_1, $\Gamma_{x_1} = (U, M_{x_1}, R_{x_1}, G)$, is weakly regular. Let $y \in M_{x_1}$, and $(r_{m+1}) \in R_{x_1}$. Suppose that:*

(1) $\forall i : 1 \leq i \leq m$, $x_{i+1} = G(x_1, (r_i)) \neq y$.

(2) $G(x_{m+1}, r_{m+1}) = G(x_1, (r_{m+1})) = y$.

(3) $y \prec (r_m)$.

Then a seed set for the weakly regular family $(U, (x_{m+1}), (r_m), G)$, is $S = \{x_1, y\} \cup \{(r_m) \backslash y\}$, and S has minimum $|M_S|$.

Proof. See Appendix C. □

Theorem 4.1 states that if a sequence of m resources,(r_m), is used to produce a sequence of $m+1$ machines, (x_{m+1}), and machine y is contained in (r_m) but does not belong to (x_{m+1}), and the resource seed set is devoid of all machines, then the machine seed set must consist of y and the first machine in (x_{m+1}). Hence, we must examine the sequences of machines generated by a matriarch when seeding weakly regular generation systems. We first present a Line Seeding (LS) sub-algorithm, before giving the general SI algorithm.

4.3 The LS Sub-Algorithm

Input: a simple path in the directed minimum spanning tree (DMST) representation of the weakly regular generation subsystem of a matriarch, x_1, that begins at x_1. The machines (vertices) in this path constitute M, and the resources (edges) used in this path constitute R. Let the path length be n.

Output: a seed set, S, for this simple path.

```
    /*We create a new line graph, known as the "seeding graph," in order to help determine a seed.*/
 1: $M_S \leftarrow \{x_1\}$.
 2: Initialize the seeding graph with one vertex, $x_1$, and zero edges.
 3: **for** $1 \leq i \leq n$ **do**
 4:     Let $y \leftarrow G(x_1, (r_i))$.
 5:     **if** $y$ is not a vertex in the seeding graph **then**
 6:         Add the machines contained in $(r_i)\backslash(x_1, y)$ that are not already on the seeding graph as new vertices. Draw a directed edge from the last vertex to the first contained machine, from the first contained machine to the second contained machine, and so on.
 7:         Add $y$ to the end of this line graph with a directed edge that comes from the last contained machine that was added.
 8:     **else**
 9:         **if** $(x_i)\backslash x_1$ and all the contained machines in $(r_i)\backslash(x_1, y)$ are *not* between the $x_1$ and $y$ vertices on the line graph (any contained machines that are not already on the seeding graph may simply be added in an appropriate position) **then**
10:             $M_S \leftarrow M_S \cup \{y\}$
11:         **end if**
12:     **end if**
13: **end for**
14: $R_S \leftarrow R\backslash M$.
15: $S \leftarrow M_S \cup R_S$.
```

At each iteration of the LS sub-algorithm, the number of intermediate machines grows by one. If a path is not strongly regular because of one of the intermediate machines, then Theorem 4.1 comes into play, and the seeding graph is a tool to indicate which machines should be added to the machine seed set.

Theorem 4.2. *The LS sub-algorithm is correct. That is, the output of the LS sub-algorithm is a seed for a simple path in the DMST of the generation subsystem of a matriarch that starts at the root of the tree.*

Proof. See Appendix C. □

4.4 Pseudocode

Input: a generation system of n machines and m resources that is weakly regular and made up of one or more disjoint families, and cost functions $J : M \to \mathbb{R}$ and $K : R \to \mathbb{R}$.

Output: a seed set, S, for this generation system.

```
1:  for all $x_i \in M$, $1 \leq i \leq n$ do
2:      Determine $\Gamma_{x_i}$.
3:  end for
    /*In the above, with each machine (vertex) as a starting point (root) in the initial generation system (directed graph), we need to find the subsystem (maximally connected subgraph) that can be generated (reached from the root). Two well known algorithms to compute the reachable components in a graph are the Breadth-First Search (BFS) and the Depth-First Search (DFS) algorithms [18, 19, 21].*/
4:  Select the $\Gamma_{x_i}$ where $|M_{x_i}| \geq |M_{x_j}|, \forall 1 \leq j \leq n$.
    /*The idea is to seed a primary subsystem first, and then go back to a secondary subsystem $M \backslash M_{x_i}$ and partition and seed iteratively.*/
5:  for all the matriarchs of the largest generation subsystem do
6:      if in the graph representation of $\Gamma_{x_i}$, $x_i$ has entering edges then
7:          Add a new vertex $x_i'$.
8:          Change these edges so that they now enter $x_i'$.
9:      end if
10:     Find the directed minimum spanning tree (DMST) in the graph of $\Gamma_{x_i}$ with root at $x_i$.
```

```
11:     $\Gamma_{x_i min} \leftarrow$ the DMST of $\Gamma_{x_i}$.
12:     for all simple paths in the DMST that begin at $x_i$ do
13:         Use the LS sub-algorithm to seed each path.
14:     end for
15:     $S_{x_i} \leftarrow \bigcup S_{path}$.
16: end for
17: Select the $S_{x_i}$ for which $\sum_{y \in M_{S_{x_i}}} J(y) + \sum_{r \in R_{S_{x_i}}} K(r)$ is a minimum.
    /*We ensure that the primary subsystem has the property that each offspring is generated from
    only one resource. Thereafter, we can select all resources to be a part of the seed set. We use
    the Chu-Liu-Edmonds algorithm [20, 26] to find the DMST for each matriarch in the primary
    subsystem. For each of these DMSTs, we can apply the DFS algorithm to find all the simple
    paths that begin at the root and utilize the LS sub-algorithm to seed each path. The seed for
    the entire subsystem for a particular DMST is the union of the seeds for each path. We pick
    the DMST seed with minimum total cost.*/
18: Remove all $x \in M_{x_i}$ from $M$.
19: if $M \neq \oslash$ then
20:     Go to Line 4.
21: else
22:     $S \leftarrow \bigcup_{x_i \in M_{♀}} S_{x_i}$.
23: end if
```

4.5 Properties

Theorem 4.3. *The SI algorithm is correct. That is, the output of the algorithm is a seed for the given generation system.*

Proof. See Appendix C. □

Theorem 4.4. *The SI algorithm is complete. That is, the algorithm will output a seed if one exists for the given generation system.*

Proof. See Appendix C. □

Theorem 4.5. *The SI algorithm is guaranteed to stop after a finite number of iterations. The time complexity for the operation of this algorithm is polynomial.*

Proof. See Appendix C. □

Theorem 4.6. *The SI algorithm produces a seed that is optimal with respect to the number of machines and the cost of the seed.*

Proof. See Appendix C. □

The assumption of disjoint families is required to ensure optimality of the seed resource set. Although the proposed algorithm will work for families that are not disjoint, no claims about optimality can be made. However, we conjecture that the resultant seed will be close to optimal. If the system does not possess disjoint families, then line 20 of the SI algorithm would need to iterate from line 1 instead of line 4.

Of the difficulties listed in Section 1.3, (b), (c), (d) and (e) are effectively handled with the SI algorithm, and (a) is ineffectively handled.

4.6 Example Application

Because the SI and RSI algorithms are so similar, the reader is referred to Chapter 3 for an example of a robotic self-replicating system that may be seeded with the SI algorithm. The example in this section serves only to illustrate the working of lines 5 through 17 of the SI algorithm. To demonstrate the applicability to *any* self-reproducing system, not just robotic ones, we use the naturally occurring atmospheric ozone cycle attacked by chlorine [14] as presented in Section 3.5.

Every machine in the cycle is a matriarch for the family, and so a seeding graph has to be produced for each machine. If we start with x_1, then the LS sub-algorithm yields the following line graph and machine seed set:

$$\begin{tikzcd} x_1 \arrow[r] & x_2 \arrow[r] & x_3 \arrow[r] & x_5 \arrow[r] & x_4 \end{tikzcd}$$

$$M_{S_{x_1}} = \{x_1, x_5\}.$$

Similarly, starting with x_2 yields:

$$\begin{tikzcd} x_2 \arrow[r] & x_1 \arrow[r] & x_3 \arrow[r] & x_5 \arrow[r] & x_4 \end{tikzcd}$$

$$M_{S_{x_2}} = \{x_2, x_5, x_1\}.$$

Starting with x_3 yields:

$$\begin{tikzcd} x_3 \arrow[r] & x_2 \arrow[r] & x_5 \arrow[r] & x_4 \arrow[r] & x_1 \end{tikzcd}$$

$$M_{S_{x_3}} = \{x_3, x_5, x_2\}.$$

Starting with x_4 yields:

$$\begin{tikzcd} x_4 \arrow[r] & x_2 \arrow[r] & x_5 \arrow[r] & x_1 \arrow[r] & x_3 \end{tikzcd}$$

$$M_{S_{x_4}} = \{x_4, x_2\}.$$

Finally, starting with x_5 yields:

$$\begin{tikzcd} x_5 \arrow[r] & x_1 \arrow[r] & x_2 \arrow[r] & x_3 \arrow[r] & x_4 \end{tikzcd}$$

$$M_{S_{x_5}} = \{x_5\}.$$

We let $\sum_{y \in M_{S_{x_i}}} J(y) := |M_{S_{x_i}}|$. The definition of K is irrelevant in this example because the sets $R_{S_{x_i}} = R \backslash M$ are equal for all $1 \leq i \leq 5$. Hence, for the cycle in Fig. 3.3, we select the seed where

$$\begin{aligned} M_S &= \{x_5\}, \text{ and} \\ R_S &= \{r_1, r_2 \backslash x_1, r_3\}. \end{aligned}$$

From an environmental standpoint, it is interesting to note how vital chlorine is to the cycle so that it always shows up in the machine seed set in one form or another.

Chapter 5

Conclusions and Future Work

Three novel algorithms to identify a seed for certain classes of generation systems have been proposed. These algorithms possess the following capabilities to various degrees: handling multiple disjoint or intersecting generation subsystems; considering resources and their composition; dealing with machines of deficient rank that are used as resources; isolating seed machines from generation cycles; and overcoming the difficulty of seeding self-reproducing systems where the generation of a machine depends on the assistance of its offspring.

The avenues for future research include examining how one can control a generation system to produce an optimal seed. Once issues of control have been resolved, the ideal of finding a seed that can initiate an evolving self-reproducing system needs to be pursued. With the theory in place to analyze generation systems, the next step is to develop theory to synthesize generation systems.

All the algorithms presented here need to be extended to 1) allow for the determination, whenever possible, of a seed of pre-specified order; 2) incorporate some notion of the quantity of a seed resource needed to perpetuate a system; and 3) recognize and compensate for time constraints that may impose a larger-size seed upon the system.

Appendix A

Product Manufacturing Cycle Model

Here, we describe a model of the product manufacturing cycle. This model demonstrates how the production of copies of the merchandise require the assistance of entities that are affected by the product. Let us assume the existence of a company that sells this product, and model the effect that this product has on the company's departments and the customer.

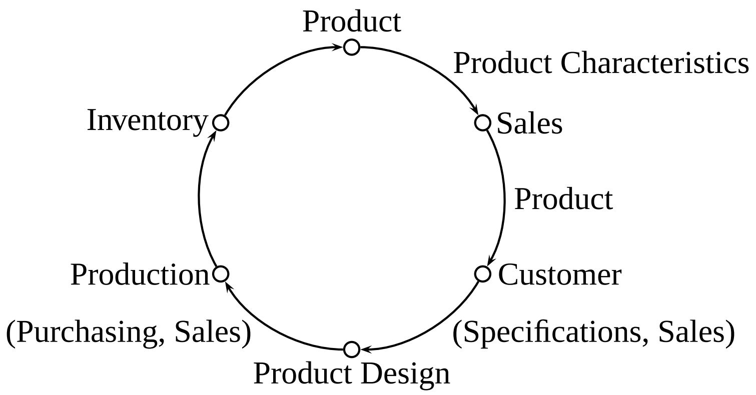

Figure A.1: Directed graph representation of a product manufacturing cycle.

The company's sales department is directly affected by the merchandise because sales are driven by the product's characteristics. Sales personnel are responsible for matching the product with potential customers. Customers, through specification documents and the number of purchases, impact future iterations of the product's design. Manufacturing production is dependent on this design, and on the purchasing of parts via monies generated from sales. Production creates inventory, from which an instance of the product is selected for sale. This cycle is illustrated in the Fig. A.1.

As the diagram indicates, it is clear that the cycle has “resources” that contain entities constituting a different stage of the cycle. In the context of the theory presented in this report, this product manufacturing cycle model is an example of a weakly regular generation system.

Appendix B

RSI Algorithm Proofs

Theorem 3.1.

Proof. The proof is by construction. Specifically, we outline an iterative algorithm that is guaranteed to identify a matriarch for a family. At the end of every iteration, the algorithm produces a partition of the family into a candidate matriarch, a set of descendants of that candidate matriarch, and a set of machines yet to be considered. During each iteration, the size of the set of machines yet to be considered is decreased by at least one unit, the size of the set of descendants of the candidate matriarch is increased by at least one unit, and the candidate matriarch itself may be updated. The algorithm terminates when the set of machines to be considered is empty, at which time the candidate matriarch is confirmed as a matriarch.

To initialize the algorithm, consider two arbitrary machines x and y of the family. Since x and y are in the family, they have a common ancestor z. We consider three cases:

(1) If $z = x$, then the candidate matriarch is x, the set of descendants of the candidate matriarch is the set of all machines obtained in the process of generating y from x (including y), and the initialization is complete.

(2) If $z = y$, then the candidate matriarch is y, the set of descendants of the candidate matriarch is the set of all machines obtained in the process of generating x from y (including x), and the initialization is complete.

(3) If z is neither x nor y, then the candidate matriarch is z, the set of descendants of the candidate is the set of all machines obtained in the process of generating both x and y from z (including x and y), and the initialization is complete.

Once the algorithm is initialized, each iteration proceeds as follows. Let x be the candidate matriarch, and consider an arbitrary machine y in the set of machines yet to be considered. Since x and y are in the family, they have a common ancestor z. We consider four cases:

(1) If $z = x$, then the candidate matriarch remains x, and all the machines obtained in the process of generating y from x (including y) are transferred into the set of descendants of the candidate matriarch and removed from the set of machines yet to be considered. This completes the iteration.

(2) If $z = y$, then the candidate matriarch becomes y, and all the machines obtained in the process of generating x from y (including x) are transferred into the set of descendants of the candidate matriarch and removed from the set of machines yet to be considered. This completes the iteration.

(3) If z is neither x nor y but is in the set of descendants of the candidate matriarch, then the candidate matriarch remains x, and all the machines obtained in the process of generating y from z (including y) are transferred into the set of descendants of the candidate matriarch and removed from the set of machines yet to be considered. This completes the iteration.

(4) If z is neither x nor y but is in the set of machines yet to be considered, then the candidate matriarch becomes z, and all the machines obtained in the process of generating both x and y from z (including x and y) are transferred into the set of descendants of the candidate matriarch and removed from the set of machines yet to be considered. This completes the iteration.

□

Theorem 3.2.

Proof. Weak connectivity of the directed graph representation of $\Gamma = (U, M, R, G)$ follows directly from the definition of a family. Indeed, since Γ is a family, for a particular $(x, y) \in M, \exists z \in M$, and $(r_n), (r_m) \in R$ such that $x = G(z, (r_n))$ and $y = G(z, (r_m))$. In the directed graph representation of Γ, there is a path from z to x through the sequence of edges labeled (r_n), and a path from z to y through the sequence of edges labeled (r_m). Hence, in the undirected version of this directed graph, there is a path from x to y via z. By the definition of weak connectivity, this means that x and y are weakly connected in the directed graph. Since this is true for all vertex pairs in the directed graph representation of a family, the entire graph is weakly connected. $\square$

Theorem 3.3.

Proof. This proof is by contradiction. Let $|M_S|$ be a minimum.

Case 1: $|M_\infty| > 0$.

Suppose that $\exists x \in M_S$ such that $\rho(x) < \rho$. From Generation Theory [5], since Γ is a family, $\exists y \in M, r \in R$ such that $x = G(y, r)$, and $\rho(x) < \rho(y) \leq \rho$.

From the definition of a seed (Definition 3.1), $\exists z \in M_S, (r_n) \in R_S$ such that $y = G(z, (r_n))$.

Thus, both z and x belong to M_S.

Let $(r_m) := ((r_n), r)$, so that $(r_m) \in R$.

Then $G(z, (r_m)) = x$, and so $S' = (M_S \backslash \{x\}) \cup R_S$ is a valid seed.

But $|M_S \backslash \{x\}| < |M_S|$, and so $|M_S|$ is not a minimum, a contradiction.

Case 2: $|M_\infty| = 0$.

Suppose that $\exists x \in M_S$ such that $\rho(x)$ is not the maximum over all machines in the family. From Generation Theory [5], since Γ is a family, $\exists y \in M, r \in R$ such that $x = G(y, r)$, and $\rho(x) < \rho(y) < \rho$.

From the definition of a seed (Definition 3.1), $\exists z \in M_S, (r_n) \in R_S$ such that $y = G(z, (r_n))$.

Thus, both z and x belong to M_S.

Let $(r_m) := ((r_n), r)$, so that $(r_m) \in R$.

Then $G(z, (r_m)) = x$, and so $S' = (M_S \backslash \{x\}) \cup R_S$ is a valid seed.

But $|M_S \backslash \{x\}| < |M_S|$, and so $|M_S|$ is not a minimum, a contradiction.

□

Theorem 3.4.

Proof. This proof follows directly from the definitions of strong regularity and generation subsystem. Indeed, if $R_{S_x} \cap M_x = \oslash$, then we need to have $|M_{S_x}| \geq 1$ so that at least one machine is present to generate the system. Since Γ_x is the generation subsystem of x, x can generate every machine in M_x by definition. Since Γ_x is strongly regular, any resources that contain machines cannot be used to generate those machines, by definition. This implies that machines additional to x are not needed. Therefore, the set $S_x = \{x\} \cup (R_x \backslash M_x)$ is a valid seed.

Moreover, $|M_{S_x}| = 1$, the minimum possible. □

Theorem 3.5.

Proof. We have to prove that the output set S is a seed for the initial self-reproducing system. Since Γ is a union of families, and $S = \bigcup S_x$ for x belonging to the set of matriarchs $M_{♀}$, it suffices to prove that each S_x is a seed for one of the constituent families. Thus, we will show that each of Steps 1 through 4 is correct.

Step 1.

By assumption, the generation system to be seeded is made up of one or more strongly regular families. The directed graph representation of a single family is weakly connected. Thus, the directed graph representation of the initial generation system is made up of one or more weakly connected components.

Each vertex in the directed graph representation belongs to a weakly connected component. Both the BFS or DFS algorithms are able to correctly find the vertices reachable from a root in a weakly connected directed graph [18]. Thus, the use of either of these algorithms ensures that this step is correct.

Step 2.

Here, the SI algorithm considers a finite number of sets each with finite cardinality. There are several known algorithms that are able to correctly count the elements in a set and sort the sets in descending order. The use of any of these algorithms results in the selection process being correct.

Step 3.

To find the directed minimum spanning tree for the selected weakly connected component requires use of the Chu-Liu-Edmonds algorithm, or Tarjan's efficient implementation of the same. These algorithms have been proved to be correct [22, 23, 26]. Just as in Step 2, there are known algorithms for correctly evaluating the sum of a functional on the elements of a set, sorting these sums, and picking the set with the minimum sum. The use of any of these algorithms results in the selection process being correct.

Step 4.

This part of the proof is similar to the proof of Theorem 3.4. Since Γ_{xmin} is the generation subsystem of x with the added property that each offspring is generated from only one resource, x can generate every machine in $M_{xmin} = M_x$ by definition. Since Γ_{xmin} is strongly regular, any resources that contain machines cannot be used to generate those machines, by definition. This implies that machines additional to x are not needed. Thus, the set $S_x = \{x\} \cup (R_{xmin} \backslash M_x)$ is a valid seed.

Therefore, S_x is a seed for all Γ_x.

□

Theorem 3.6.

Proof. We have to show that if a seed exists, the algorithm in this paper will output one possible seed. Consider that a seed for a generation system always exists - this is the trivial seed, consisting of all the machines and resources in the generation system, i.e., $S = M \cup R$. Indeed, the algorithm presumes this seed at the start, before removing redundant resources and machines that belong to a matriarch's subsystem. Theorem 3.5 shows that the output of the algorithm is a seed.

Thus, completeness is guaranteed. □

Theorem 3.7.

Proof. Note that each iteration of the algorithm removes elements from a set with finite cardinality, and the algorithm stops once the set is depleted.

Consider the time complexity of Steps 1 to 5 during the first iteration of the algorithm.

In Step 1, the use of either one of the BFS or DFS algorithms has time complexity $O(n+m)$ [18].

In Step 2, the fact that each machine has to be visited in order to determine the cardinality of the machine set of its generation subsystem results in a time complexity of $O(n)$.

In Step 3, the time complexity of the DMST algorithm is $O(n_p m_p)$ [20], where n_p is the number of machines in the primary subsystem, and m_p is the number of resources in the primary subsystem. Accounting for the possibility that there is more than one matriarch to apply the DMST algorithm to, and that the cost of the seed for each matriarch's subsystem needs to be evaluated, the time complexity of this step is $O(n_♀ n_p m_p + n_♀)$, where $n_♀$ is the number of matriarchs.

In Step 4, the fact that (in the worst case) all primary subsystem resources have to be visited in order to remove any contained machines results in a time complexity of $O(m_p)$.

In Step 5, all primary subsystem machines have to be removed from the original machine set, so that the time complexity of this step is $O(n_p)$.

Thus the overall time complexity of Steps 1 to 5 during the first iteration of the algorithm is $O(2n + m + n_p + m_p + n_♀ n_p m_p + n_♀)$.

In the best case, the SI algorithm executes once and there is only one matriarch. This implies that $n_p = n$, $m_p = m$ and $n_♀ = 1$, so that the resultant best-case time complexity is $O(3n + nm + 2m + 1)$.

In the worst case, either the SI algorithm executes once and there are n matriarchs, or the SI algorithm executes n times and each machine is a matriarch for a family that has a singleton set of machines. Thus, we have two possibilities to consider.

The first possibility is $n_p = n$, $m_p = m$ and $n_♀ = n$, so that the resultant time complexity is $O(n^2 m + 4n + 2m)$.

The second possibility is $n_p = 1$ and $m_p = 1$ implying that $n_{\female} = 1$, and after the first pass through the algorithm, Steps 2-5 are repeated $n - 1$ times. This time complexity is $O(2n + m + 4) + O((n-1)(n+4)) = O(n^2 + 5n + m)$. □

Theorem 3.8.

Proof. Let $\Gamma = (U, M, R, G)$ be made up of k strongly regular disjoint families. From Theorem 3.1, there are at least k matriarchs. Since each family is the generation subsystem of a matriarch, and each of these subsystems is strongly regular, Theorem 3.4 indicates that the minimum $|M_S|$ is k.

In the proof of Theorem 3.5, we have shown that each pass through Steps 1 to 4 of the SI algorithm produces a seed for a family, before the family is removed from the original generation system. This seed for the family contains one machine. If there are k families in the original system, the SI algorithm will iterate k times before returning a seed that is the union of the seed sets for each family. Thus, there will be k machines in M_S.

Therefore, the number of machines is optimal because it is the minimum it could be.

By assumption, all the machines in the given generation system need to be produced. Hence, the optimal seed for each family must include the least costly resources such that all machines in the family are generated. This implies that there must exist a path between the root vertex and all other vertices in the directed graph representation of the subsystem of a matriarch, and the DMST that is found via the Chu-Liu-Edmonds algorithm satisfies this property with minimal cost. If there are multiple matriarchs, the resource set that is selected is the least costly. Taking all such minimal cost resources produces an optimal seed resource set for each family, and since the families are disjoint, the union of these sets result in a seed that is optimal with respect to the cost of the resources. □

Theorem 3.9.

Proof. As a result of expanding M or R, the rank of a family will either increase or stay constant. This is because there are now more machines and resources in the generation system, and so it is possible that machines originally located in the outer layer are now able to produce offspring. Hence, it is possible that the rank increases.

Consider the original family, Γ, prior to the expansion of M or R. Let x be a machine in the outer layer of Γ. Since generation always proceeds outwards [5] and Γ is a family, expanding M or R may result in an increase in the rank of the system. If this occurs, there is now a degenerate machine y that is a descendant of x. In other words, now $\exists (r_n) \in R$ such that $G(x, (r_n)) = y$.

Since y is degenerate, it does not need to belong to M_S, so that $|M_S|$ remains the same. Also, if (r_n) uses resources that already belong to R_S, then $|R_S|$ stays unchanged.

However, if (r_n) uses resources that differ from those in R_S, then these resources need to be added to the resource seed set. Hence, $|R_S|$ increases, producing a corresponding increase in $|S|$. $\square$

Appendix C

SI Algorithm Proofs

Theorem 4.1.

Proof. This proof follows directly from the definition of a seed. First, we are given that $y \prec (r_m)$. Since $\forall 1 \leq i \leq m$, $x_{i+1} = G(x_1, (r_i))$, a seed for the weakly regular family $(U, (x_{m+1}), (r_m), G)$ is:

$$\begin{aligned} S &= \{x_1\} \cup \{(r_m)\}; \\ &= \{x_1, y\} \cup \{(r_m) \backslash y\}. \end{aligned}$$

If $R_S \cap M_S = \oslash$, then we need to have $|M_S| \geq 1$ so that at least one machine is present to generate the system. We are given that x_1 can produce every machine in (x_{m+1}) using (r_m). From the seed set S above, since (r_m) contains only y but y cannot be generated by x_1, the system needs to be started with both x_1 and y. Therefore, $|M_S| = 2$, the minimum possible. □

Theorem 4.2.

Proof. This proof uses mathematical induction. We assume that the generation subsystem of our matriarch x_1, where $\Gamma_{x_1} = (U, M_{x_1}, R_{x_1}, G)$, is a DMST, and that we have selected a simple path in this tree that starts at x_1. Let M be the set of machines that are the vertices in this path, and R be the set of resources that are the edges in this path. Let $R_S = R \backslash M$, and $M_S = \{x_1\}$.

Let r be the first resource edge in this path, and s be the second resource edge in this path. Let $y := G(G(x_1, r), s)$, which is different from $G(x_1, r)$ by the definition of a path.

Consider $G(x_1, r)$. If $r \succ y$, the sub-algorithm takes $M_S = M_S \cup \{y\}$, and by Theorem 4.1, the new M_S forms a seed for the path when united with R_S. Otherwise, the original M_S is still a seed when united with R_S, since y is not required.

For the induction hypothesis, assume that M_S forms a seed with R_S when x_1 uses a sequence of (r_{k-1}) resources. Let $y := G(G(x_1, (r_{k-1})), r_k)$, which is different from $G(x_1, (r_{k-1}))$ by the definition of a path.

Consider $G(x_1, (r_{k-1}))$. If $(r_{k-1}) \succ y$, the sub-algorithm takes $M_S = M_S \cup \{y\}$, and by Theorem 4.1, the new M_S forms a seed for the path when united with R_S. Otherwise, the original M_S is still a seed when united with R_S, since y is not required. $\square$

Theorem 4.3.

Proof. We have to prove that the output set S is a seed for the initial self-reproducing system. Since Γ is a union of families, and $S = \bigcup S_x$ for x belonging to the set of matriarchs $M_{♀}$, it suffices to prove that each S_x is a seed for one of the constituent families. Thus, we will show that each of Steps 1 through 3 is correct.

Step 1.

By assumption, the generation system to be seeded is made up of one or more weakly regular families. The directed graph representation of a single family is weakly connected. Thus, the directed graph representation of the initial generation system is made up of one or more weakly connected components.

Each vertex in the directed graph representation belongs to a weakly connected component. Both the BFS or DFS algorithms are able to correctly find the vertices reachable from a root in a weakly connected directed graph [18]. Thus, the use of either of these algorithms ensures that this step is correct.

Step 2.

Here, the SI algorithm considers a finite number of sets each with finite cardinality. There are several known algorithms that are able to correctly count the elements in a set and sort the sets in descending order. The use of any of these algorithms results in the selection process being correct.

Step 3.

To find the directed minimum spanning tree for the selected weakly connected component requires use of the Chu-Liu-Edmonds algorithm, or Tarjan's efficient implementation of the same. These algorithms have been proved to be correct [22, 23, 26]. We have shown by Theorem 4.2 that the LS sub-algorithm is correct for any path in the tree. Since the union of seed sets is itself a seed set, $S_x = \bigcup S_{path}$ is a valid seed. Just as in Step 2, there are known algorithms for correctly evaluating the sum of a functional on the elements of a set, sorting these sums, and picking the set with the minimum sum. The use of any of these algorithms results in the selection process being correct.

Therefore, S_x is a seed for all Γ_x. □

Theorem 4.4.

Proof. We have to show that if a seed exists, the algorithm in this paper will output one possible seed. Consider that a seed for a generation system always exists - this is the trivial seed, consisting of all the machines and resources in the generation system, i.e., $S = M \cup R$. Indeed, the algorithm presumes this seed at the start, before removing redundant resources and machines that belong to a matriarch's subsystem. Theorem 4.3 shows that the output of the algorithm is a seed.

Thus, completeness is guaranteed. □

Theorem 4.5.

Proof. The LS sub-algorithm is convergent because no circuits exist in the DMST, and there are a finite number of paths of finite length that begin at the root of the tree. Each iteration of the SI algorithm removes elements from a set with finite cardinality, and this algorithm stops once the set is depleted. Consider the time complexity of Steps 1 to 4 during the first iteration of the algorithm.

In Step 1, the use of either one of the BFS or DFS algorithms has time complexity $O(n+m)$ [18].

In Step 2, the fact that each machine has to be visited in order to determine the cardinality of the machine set of its generation subsystem results in a time complexity of $O(n)$.

In Step 3, the time complexity of the DMST algorithm is $O(n_p m_p)$ [20], where n_p is the number of machines in the primary subsystem, and m_p is the number of resources in the primary subsystem. The use of the DFS algorithm to identify the simple paths in the DMST has time complexity $O(n+m)$. The LS sub-algorithm visits all the machines in a simple path once, and this is repeated for a finite number of simple paths. The fact that (in the worst case) all primary subsystem resources have to be visited in order to remove any contained machines results in a time complexity of $O(m_p)$. Accounting for the possibility that there is more than one matriarch to apply the DMST algorithm to, this entire step could be repeated $n_♀$ times, where $n_♀$ is the number of matriarchs. Thus this step has polynomial time complexity.

In Step 4, all primary subsystem machines have to be removed from the original machine set, so that the time complexity of this step is $O(n_p)$.

Thus the overall time complexity of Steps 1 to 4 during the first iteration of the algorithm is of polynomial order. □

Theorem 4.6.

Proof. Let $\Gamma = (U, M, R, G)$ be made up of k weakly regular disjoint families. From Theorem 3.1, there are at least k matriarchs. Since each family is the generation subsystem of a matriarch, the minimum $|M_S|$ is k.

In the proof of Theorem 4.3, we have shown that each pass through Steps 1 to 3 of the SI algorithm produces a seed for a family, before the family is removed from the original generation system. By Theorem 4.1, this seed contains the minimum number of machines to generate each path. For paths with a common sub-path, the minimum number of machines to generate the common sub-path is the same and is unaffected by the union operation. For disjoint paths, the minimum number of machines to generate the paths is the sum of the minimum number of machines to generate each path, which is the number of machines produced by the union operation. Thus, the number of machines in the machine seed set is a minimum for each family. If there are k disjoint families in the original system, the SI algorithm will iterate k times before returning a seed that is the union of the seed sets for each family. Therefore, the number of machines is optimal because it is the minimum it could be.

By assumption, all the machines in the given generation system need to be produced. Hence, the optimal seed for each family must include the least costly resources such that all machines in the family are generated. This implies that there must exist a path between the root vertex and all other vertices in the directed graph representation of the subsystem of a matriarch, and the DMST that is found via the Chu-Liu-Edmonds algorithm satisfies this property with minimal cost. If there are multiple matriarchs, the seed set that is selected is the least costly. Taking all such minimal cost seeds produces an optimal seed set for each family, and since the families are disjoint, the union of these sets results in a seed that is optimal with respect to cost. $\square$

Bibliography

[1] J. von Neumann, *Theory of Self-Reproducing Automata*, A. Burks, Ed. University of Illinois Press, 1966.

[2] R. A. Freitas Jr. and R. C. Merkle, *Kinematic Self-Replicating Machines*. Landes Bioscience, 2004.

[3] M. Sipper, “Fifty years of research on self-replication: An overview,” *Artifical Life*, vol. 4, no. 3, pp. 237–257, 1998.

[4] P. Owens and A. G. Ulsoy, “Self-replicating machines: Preventing degeneracy,” The University of Michigan, Tech. Rep. CGR-06-02, 2006.

[5] P. Kabamba, “The von Neumann threshold of self-reproducing systems: Theory and computation,” The University of Michigan, Tech. Rep. CGR-06-11, 2006.

[6] A. Menezes and P. Kabamba, “Information requirements for self-reproducing systems in lunar robotic colonies,” in *Proceedings of the 57th International Astronautical Congress*, no. IAC-06-A5.P.04, 2-6 October 2006.

[7] ——, “A combined seed-identification and generation analysis algorithm for self-reproducing systems,” in *Proceedings of the 2007 American Control Conference*, 11-13 July 2007, pp. 2582–2587.

[8] ——, “An optimal-seed identification algorithm for self-reproducing systems,” in *Proceedings of the 58th International Astronautical Congress*, no. IAC-07-D3.2.02, 24-28 September 2007.

[9] ——, “Optimal seeding of a class of self-reproducing systems,” in *Submitted to the 17th IFAC World Congress*, 6-11 July 2008.

[10] B. Foing, “Roadmap for robotic and human exploration of the moon and beyond,” in *Proceedings of the 56th International Astronautical Congress*, no. IAC-05-A5.1.01, 17-21 October 2005.

[11] J. R. Wertz and W. J. Larson, Eds., *Space Mission Analysis and Design*, 3rd ed. Microcosm Press, 1999.

[12] R. Diestel, *Graph Theory*, 3rd ed. Springer-Verlag Heidelberg, 2005.

[13] H. Lodish, A. Berk, P. Matsudaira, C. A. Kaiser, M. Krieger, M. P. Scott, L. Zipursky, and J. Darnell, *Molecular Cell Biology*, 5th ed. W. H. Freeman, 2004.

[14] P. V. Hobbs, *Introduction to Atmospheric Chemistry*. Cambridge University Press, 2000.

[15] W. H. Adey and K. Loveland, *Dynamic Aquaria*, 2nd ed. Academic Press, 1998.

[16] G. S. Chirikjian, Y. Zhou, and J. Suthakorn, “Self-replicating robots for lunar development,” *IEEE/ASME Transactions on Mechatronics*, vol. 7, no. 4, Dec. 2002.

[17] J. Suthakorn, Y. T. Kwon, and G. S. Chirikjian, “A semi-autonomous replicating robotic system,” in *Proceedings of the 2003 IEEE International Symposium on Computational Intelligence in Robotics and Automation*, Jul. 2003.

[18] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. The MIT Press, 2001.

[19] A. V. Aho, J. E. Hopcraft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*. Addison-Wesley Publishing Company, 1974.

[20] S. Even, *Graph Algorithms*. Computer Science Press, 1979.

[21] J. Hopcroft and R. Tarjan, "Algorithm 447: Efficient algorithms for graph manipulation," *Communications of the ACM*, vol. 16, no. 6, pp. 372–378, 1973.

[22] Y. J. Chu and T. H. Liu, "On the shortest arborescence of a directed graph," *Scientia Sinica*, vol. 14, pp. 1396–1400, 1965.

[23] J. Edmonds, "Optimum branchings," *Journal of Research of the National Bureau of Standards*, vol. 71B, no. 4, pp. 233–240, 1967.

[24] F. Bock, "An algorithm to construct a minimum directed spanning tree in a directed network," *Developments in Operations Research*, pp. 29–44, 1971.

[25] R. M. Karp, "A simple derivation of Edmonds' algorithm for optimum branchings," *Networks*, vol. 1, pp. 265–272, 1971.

[26] R. E. Tarjan, "Finding optimum branchings," *Networks*, vol. 7, pp. 25–35, 1977.