Optimal Seeding of Self-Reproducing Systems

Amor A. Menezes*,**
University of California, Berkeley

Pierre T. Kabamba†
University of Michigan

Abstract This article is motivated by the need to minimize the number of elements required to establish a self-reproducing system. One such system is a self-reproducing extraterrestrial robotic colony, which reduces the launch payload mass for space exploration compared to current mission configurations. In this work, self-reproduction is achieved by the actions of a robot on available resources. An important consideration for the establishment of any self-reproducing system is the identification of a *seed*, for instance, a set of resources and a set of robots that utilize them to produce all of the robots in the colony. This article outlines a novel algorithm to determine an optimal seed for self-reproducing systems, with application to a self-reproducing extraterrestrial robotic colony. Optimality is understood as the minimization of a cost function of the resources and, in this article, the robots. Since artificial self-reproduction is currently an open problem, the algorithm is illustrated with a simple robotic self-replicating system from the literature and with a more complicated self-reproducing example from nature.

Keywords
Combinatorial optimization, discrete mathematics, graph algorithm applications, seed identification, self-reproducing systems, space robotics

1 Introduction

1.1 Motivation

Current phased approaches to space colonization see the development of an enduring extraterrestrial robotic presence. Several space agency road maps, of which [17] is typical, suggest that individual countries will deploy advanced robots as needed 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, 80 additional units of mass are required to be launched as well [49]—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. Extraterrestrial 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 solution to the problem of establishing extraterrestrial colonies.

To minimize the payload mass for this kind of self-reproducing robotic system, it would be even more efficient to identify the required elements for the initiation of the system and send the smallest

* Corresponding author.
** Department of Bioengineering, University of California, Berkeley, Mailcode 3220, Stanley Hall, Room 176, Berkeley, CA 94720. E-mail: amenezes@berkeley.edu
† Department of Aerospace Engineering, University of Michigan, 1320 Beal Avenue, Ann Arbor, Michigan 48109-2140. E-mail: kabamba@umich.edu

number of these elements into space. Indeed, this practical need to minimize the number of elements required for system establishment is important for any self-reproducing system, and is especially important for large self-reproducing systems with the capacity to evolve and adapt while using fixed, unchangeable reproduction rules (see [33] for examples of these systems). References [31] and [32] provide algorithms that identify a seed for various classes of self-reproducing systems. This work builds on these results by providing a seeding algorithm that is applicable to a more general class of self-reproducing systems. The proposed algorithm identifies a minimal seed, and is more generally applicable because it is able to seed self-reproducing systems where a progenitor uses some of its progeny as resource to self-reproduce. Such systems are prevalent; examples include the Krebs cycle in a cell [30], the atmospheric ozone cycle with and without attack by chlorine [24], and the nitrogen cycle when starting a new aquarium [1]. We cite natural examples instead of robotic examples here because (1) there are few instances of robotic self-reproduction in the literature, due to the difficulties posed by unstructured environments [6], and (2) it is anticipated that engineered self-reproducing systems will resemble those found in nature. Hence, the goal of this article is the development of an algorithm that is capable of optimally seeding self-reproducing systems, including those where the self-reproduction of a parent uses its offspring as a resource.

1.2 Literature on Self-Reproducing Systems

The seminal work of John von Neumann [47] prompted the extensive study of self-reproducing systems, including cellular automata, computer programs, kinematic machines, molecular machines, and robotic colonies. A comprehensive overview of this field is documented in [19, 40]. In a landmark conceptual study on a self-replicating lunar factory [18], a system that included paving, mining, casting, and mobile assembly and repair robots was proposed. Inspired by this work, Chirikjian et al. [7] suggested a factory system composed of self-replicating multifunctional robots that could mine and transport materials and components within a lunar manufacturing facility. The work also demonstrated the feasibility of a self-replicating robot with a prototype made of LEGO Mindstorms components. At the same time (and in the years since), a number of researchers have developed modular self-replicating, self-assembling, and/or self-reconfigurable robots (see, e.g., [5, 8, 20–23, 26, 27, 29, 36, 38, 45, 50, 53]). A current survey of the state of the art and the challenges facing modular, self-reconfigurable robot systems is given in the Grand Challenges of Robotics article [51] and in [35, 41]. Other reviews are also available [10, 16, 37].

As the references above and those therein indicate, the focus has shifted to provable control of the modules of a single self-reconfigurable robot—the realization of various topologies [21], efficient and distributed control of a large number of modules [3, 48], recovery from module failures [52], and even module self-repair [9, 42]. Approaches for local control include reinforcement learning [46], cellular automata [4], and hormone-inspired swarming for self-organization [39]. This shift in focus to local control is due, in part, to the difficulty of achieving artificial-system self-reproduction in unstructured environments [6].

By virtue of the harsh environment an extraterrestrial robotic colony operates in, self-reproducing robots need to learn, adapt, and possibly evolve to be tolerant of external disturbances that can affect the collective's overall goals (see [33]). Recently [34], we examined the performance of a system consisting of multiple mining and ore-processing robots, where each individual robot is also capable of self-reproduction. It is such a system of multiple self-reproducing robots, modeled at a conceptual level, that is under consideration in this work.

1.3 Outline of This Article

Section 2 highlights a theory of self-reproducing systems, discusses what makes the general seeding problem difficult, and presents relevant results of limited solutions to seeding available in the literature. Section 3 details the necessary definitions and assumptions for seed identification, outlines a seed identification (SI) algorithm, and analyzes its properties. Section 4 illustrates the application of the algorithm to self-replicating systems documented in the literature. Section 5 presents conclusions.

2 Background

2.1 Highlights of Generation Theory

The theoretical framework of this article is *generation theory* [28], which formalizes self-reproduction by *machines*, a term describing any entity that is capable of producing an offspring, regardless of its physical nature. A robot, a bacterium, or even a piece of software code is considered to be a machine in this theory if it can produce another robot, another 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. 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 that are neither machines nor resources.
- $M \subseteq U$ is a *set of machines*, $|M| \geq 1$.
- $R \subseteq U$ is a *set of resources* that can each be utilized for self-reproduction. Each resource is an ordered list of elements. A resource ordered list can include machines and elements of other resource ordered lists.
- $G : M \times R \rightarrow U$ is a *generation function* that maps a machine and a resource ordered list into an outcome in the universal set.

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

$$y = G(x, r). \tag{1}$$

In (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 make use of concepts from graph theory [13]. Equation (1) may be represented by a *directed reproduction graph*, γ, as shown in Figure 1. In this diagram, the machine x and the outcome y are vertices, the resource ordered list r is an edge, and the direction of the edge indicates that it is the machine x that uses the resource ordered list r to generate the outcome y.

Definition 2. The *directed-graph representation* of a generation system $\Gamma = (U, M, R, G)$ is the directed supergraph containing all directed reproduction graphs that produce outcomes in M.

A sample directed graph representation of a generation system is depicted in Figure 2.

Let $(r_\mu) = r_1, r_2, \ldots, r_\mu$ be a sequence of μ resource ordered lists from R. We define the notation

$$G(x, (r_\mu)) := G(\ldots G(G(x, r_1), r_2) \ldots, r_\mu) \tag{2}$$

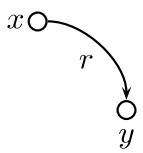


Figure 1. An example of a directed reproduction graph, which indicates that the machine x is capable of generating the outcome y using the resource ordered list r, that is, $y = G(x, r)$.

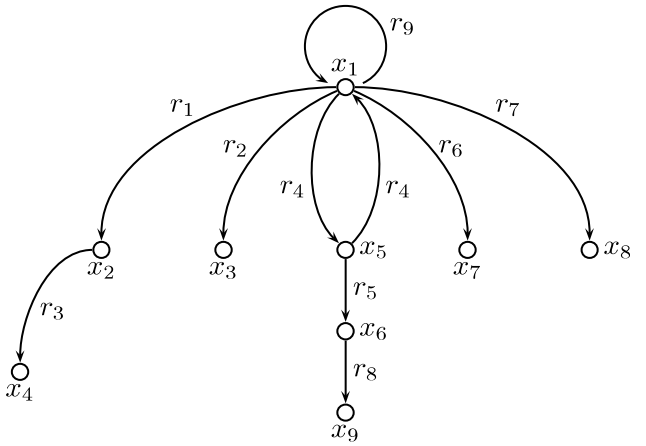


Figure 2. An example of a directed-graph representation of a generation system, where $M = \{x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9\}$ and $R = \{r_1, r_2, r_3, r_4, r_5, r_6, r_7, r_8, r_9\}$. All directed reproduction graphs for this generation system that produce outcomes in M are shown.

to denote the outcome of generation using the sequence (r_μ)—see Figure 3. This notation assumes that the intermediate outcomes of generation, $G(x, r_1)$, $G(G(x, r_1), r_2)$,…, $G(\ldots G(G(x, r_1), r_2)\ldots, r_{\mu-1})$, are all machines. The sequence of machines thus generated is called the *lineage* of x through the sequence (r_μ). In the trivial case where $\mu = 0$, we define $G(x, (r_0)) = x$, that is, a sequence of resource ordered lists of length zero yields a replication.

Definition 3. A *generation subsystem* of a generation system $\Gamma = (U, M, R, G)$ is a quadruple $\Gamma_1 = (U_1, M_1, R_1, G|_{M_1 \times R_1})$ such that

(1) $U_1 \subseteq U$, $M_1 \subseteq M$, $R_1 \subseteq R$, and

(2) $\Gamma_1 = (U_1, M_1, R_1, G|_{M_1 \times R_1})$ is itself a generation system.

An example of a generation subsystem of the generation system in Figure 2 is specified by $\Gamma_1 = (U_1, M_1, R_1, G|_{M_1 \times R_1})$ where $U_1 = M_1 \cup R_1$, $M_1 := \{x_5, x_6, x_9\}$, $R_1 := \{r_5, r_8\}$, and $G|_{M_1 \times R_1}$ states that $x_6 = G(x_5, r_5)$ and $x_9 = G(x_6, r_8)$. The machine sequence x_6, x_9 is the lineage of x_5 through the resource-ordered-list sequence r_5, r_8.

We formally define a seed in Section 3. Intuitively, since self-reproduction is achieved by the actions of a machine on available resource ordered lists, a seed for a self-reproducing system consists of a set of machines and a set of resource ordered lists such that all of the machines in the generation system are produced from a seed machine processing a finite sequence of seed resource ordered lists.

Current state-of-the-art self-reproducing systems are relatively simple, due to the nascent stage of the technology, and present-day generation systems can either be built with good seeds or be re-engineered when better seeds are desired. Yet, large and complex self-reproducing systems are envisioned for the future—systems that have the capacity to evolve and adapt to changing environmental conditions [33]. It is anticipated that design engineers will take advantage of this evolutionary capability

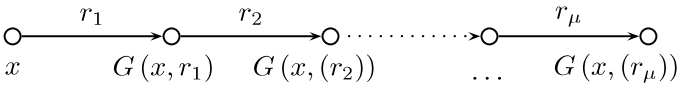


Figure 3. The directed-graph representation of a generation system where machine x initiates generation using the first resource ordered list of a sequence of μ resource ordered lists, (r_μ). The intermediate machine outcomes sequentially utilize resource ordered lists from (r_μ) to finally generate the machine $G(x, (r_\mu))$.

to evolve an appropriate configuration or functionality for a generation system prior to its deployment in an initial environment. This evolved generation system may include unnecessary or unexpected intermediate machines and resource ordered lists that are a by-product of evolution. Because the evolution of a generation system depends on the time-history of the experienced environmental conditions, which are always variable, an initially good seed may no longer be suitable or feasible for deployment, a better albeit unknown seed may exist, and re-engineering for seed suitability may be impossible due to the inherent variability in the evolutionary engineering process. Hence, a seed for the large, evolved generation system will have to be determined, and the method in this work is a precursor to more advanced techniques for seeding evolving generation systems.

2.2 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 generation subsystems, each with a different seed. Alternatively, there could be multiple, intersecting generation 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 generation system.

(b) The potential for generation cycles (sequences of generations resulting in the production of a machine identical to itself after n generations) within a given self-reproducing system. If these cycles exist, then one naturally wonders which of the machines in a particular cycle, if any, should belong to the seed. Consideration of machine and resource cost, and the consequences of (e), is required.

(c) The fact that degenerate machines (machines whose progeny will eventually no longer be machines; see [28]) 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 there is a need to identify a least-degenerate machine 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, that is, an element can belong to more than one resource list.

(e) The existence of self-reproducing systems where the generation of a copy of a machine depends on the assistance of its offspring. Typically, this phenomenon manifests itself as a combination of (b) and (d), when a resource ordered list 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 take the progenitor, its offspring, neither one, or both to belong to the seed.

It is perhaps because of all of these factors that the seeding problem is still mostly open. The only known works in this area are our previous attempts at tackling restricted versions of this problem. Reference [31] resulted in the seed identification and generation analysis (SIGA) algorithm, and [32] presented a restricted seed identification (RSI) algorithm that is applicable to a larger class of self-reproducing systems than [31].

2.3 Relevant Seeding Literature and Definitions

2.3.1 The SIGA Algorithm

In [31], we allowed each resource to be an ordered list of physical elements that could include machines. We therefore defined a containment relation as follows.

Definition 4. If machine x_i belongs to a resource ordered list 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*. We can equivalently say that r_j *contains* x_i, and we write this as $r_j \succ x_i$.

We assumed that if a machine x was contained in the resource ordered list r ($x \prec r$), then the ordered sublist of the elements of r that did not contain the machine x was also a resource ordered list, that is, $r\backslash x \in R$. We also assumed that there existed a machine in the generation system that was capable of producing any machine in the system after μ generations. The idea of the algorithm was to remove all degenerate machines from the sets M and R, select one of the remaining nondegenerate machines to be a seed machine, and select the set $R\backslash M$ to be the resource seed set. In short, of the difficulties listed in Section 2.2, (c) was effectively handled, (b) and (d) were ineffectively handled, and (a) and (e) were not handled.

2.3.2 The RSI Algorithm

Reference [32] took a more general approach to the seeding problem, also presenting necessary and sufficient conditions to find an optimal seed for a larger class of generation systems. It developed an algorithm based on the following definition.

Definition 5. The generation system $\Gamma = (U, M, R, G)$ is *strongly regular* if, whenever $y = G(x, (r_\mu))$, where x and y are machines and (r_μ) is a sequence of μ resource ordered lists, we have $y \not\prec r$ for all ordered lists r that constitute the sequence (r_μ).

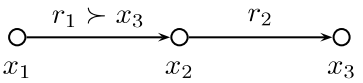
Thus, in a strongly regular generation system, if a machine is contained in a resource ordered list, then that resource ordered list cannot be utilized in any sequence of resource ordered lists used to generate the machine (Figure 4). The idea of the RSI algorithm was to separately seed specific subsets of the generation system. Of the difficulties listed in Section 2.2, (b) and (c) were effectively handled, (a) and (d) were ineffectively handled, and (e) was not handled. Determining whether a given generation system was strongly regular became an added difficulty.

2.4 SI Algorithm Overview

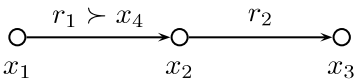
The algorithm in this article is similar to the RSI algorithm, and is inspired by genealogy. It first determines the progeny of each machine in a given generation system, picks a machine with the largest number of descendants, examines that machine's family tree to find a seed with minimum cost that generates the descendants, and iterates until all machines in the generation system are considered. The new algorithm effectively handles difficulties (b), (c), (d), and (e) in Section 2.2, and partially handles difficulty (a).

3 Seed Identification

This section formulates a seed identification problem and presents an extended version of the RSI algorithm, the SI algorithm, to solve this problem.



(a) The directed graph representation of a generation system that is not strongly regular.



(b) The directed graph representation of a strongly regular generation system.

Figure 4. As these directed-graph representations show, a resource ordered list that contains a machine cannot constitute a sequence of resource ordered lists used to generate that machine in a strongly regular generation system.

3.1 Preliminaries

We define a seed as follows.

Definition 6. Let $\Gamma = (U, M, R, G)$ be a generation system, and let $\nu \geq 1$ be a natural number. A *seed* of order $\nu\mu$ for Γ is a set

$$S = M_S \cup R_S, \tag{3}$$

where

$$M_S = x_1, x_2, \ldots, x_\nu, \qquad M_S \subseteq M, \tag{4}$$

and

$$R_S = r_1, r_2, \ldots, r_\mu, \qquad R_S \subseteq R, \tag{5}$$

such that $\forall y_1 \in M$, $\exists \mu_1 < \infty$, $\exists r_1 \in R_S$, $r_2 \in R_S$,..., $r_{\mu_1} \in R_S$ (i.e., a sequence of μ_1 resource ordered lists (r_{μ_1}) where each resource ordered list is an element of R_S), and $\exists y_0 \in M_S$ such that $G(y_0, (r_{\mu_1})) = y_1$.

That is, a seed for a generation system consists of ν machines and μ resource ordered lists such that all of the machines in the generation system can be produced from a seed machine processing a finite sequence of seed resource ordered lists. We allow $M_S \cap R_S = \emptyset$.

We can also relax the notion of strong regularity.

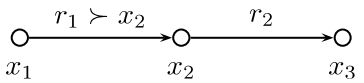
Definition 7. The generation system $\Gamma = (U, M, R, G)$ is *weakly regular* if whenever $y = G(x, r)$, where x and y are machines and r is a resource ordered list, we have $y \not\prec r$.

Thus, in a weakly regular generation system, no machine can be contained in any resource ordered list used to produce that machine (Figure 5). The difference between strong regularity and weak regularity lies in the location and number of resource ordered lists where offspring containment is allowed. In a strongly regular generation system, containment of an offspring machine is not permitted in any resource ordered list constituting a sequence of resource ordered lists used to generate that machine. In a weakly regular generation system, containment of an offspring machine is permitted in any resource ordered list constituting a sequence of resource ordered lists used to generate intermediate machines, as long as the containment does not occur with the resource ordered list that helps immediately generate the considered offspring machine. It is important to note this distinction and identify how to seed such self-reproducing systems, because we find weakly regular generation systems in nature [1, 24, 30].

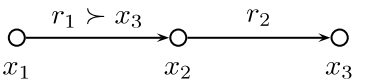
An easy consequence is the following.

Theorem 1. *Strong regularity is a sufficient condition for weak regularity.*

Proof. See Appendix. □



(a) The directed graph representation of a generation system that is not weakly regular.



(b) The directed graph representation of a weakly regular generation system.

Figure 5. As these directed-graph representations show, a resource ordered list that contains a machine cannot be utilized to generate that machine in a weakly regular generation system.

Remark 1. The converse is not true.

A formalization of a common genealogical concept is the following, which specifies that two machines that have a common ancestor are kin.

Definition 8. A *family* of a generation system $\Gamma = (U, M, R, G)$ is a generation subsystem $\Gamma_1 = (U_1, M_1, R_1, G|_{M_1 \times R_1})$ where, $\forall (x, y) \in M_1 \times M_1$, $\exists z \in M_1$, $\exists (r_n)$ with $n \geq 0$ and $r \in R_1$ for all r constituting (r_n), and $\exists (r_m)$ with $m \geq 0$ and $r \in R_1$ for all r constituting (r_m) such that $x = G(z, (r_n))$ and $y = G(z, (r_m))$. A *subfamily* is a subset of a family $\Gamma_1 = (U_1, M_1, R_1, G|_{M_1 \times R_1})$ that is itself a family.

Remark 2. A lineage is a subfamily.

It is easy to show that the notion of a family is related to the notion of a connected subgraph as follows.

Theorem 2. *The directed-graph representation of a family is weakly connected.*

Proof. See Appendix. □

However, the two notions of family and connected subgraph are distinct because the definition of the former explicitly specifies the existence of a common ancestor, but the definition of the latter does not explicitly specify reachability to a common vertex.

We can assign another genealogical term to an ancestor at the "head" of a family.

Definition 9. A *matriarch* of a family $\Gamma_1 = (U_1, M_1, R_1, G|_{M_1 \times R_1})$ is an element $x_♀ \in M_1$ such that $\forall x \in M_1$, $x \neq x_♀$, $\exists (r_\mu)$ selected from the resource ordered lists of R_1 such that $G(x_♀, (r_\mu)) = x$.

This definition leads intuitively to the following theorem, which is not trivial to prove.

Theorem 3. *Every family has a matriarch.*

Proof. See Appendix. □

3.2 Assumptions

We list the basic assumptions of our approach to seeding a given generation system.

Assumption 1. M and R are finite sets.

Assumption 2. For each $r \in R$, an inexhaustible supply is available.

For a lunar robotic colony, this statement makes the dependence on in situ resources explicit and assumes that the extraterrestrial store of resources will not run out. This assumption is consistent with generation theory, which does not specify quantities of resources or machines.

Assumption 3. All the machines in the generation system must be produced, although they need not all belong to a seed.

That is, a generation system is specified a priori, and it is this entire system that must be seeded. One reason for why an entire system must be seeded is the possibility that one or more machines are necessary for the primary (non-self-reproductive) functions of the system. Another reason is that the

given generation system will typically include intermediate machines that are needed to facilitate the self-reproduction or self-replication steps of other machines.

Assumption 4. If a machine x is contained in a resource ordered list r ($x \prec r$), then the ordered sublist of the elements of r that does not contain the machine x is also a resource ordered list, that is, $r\backslash x \in R$.

This assumption is the same as one employed by the SIGA algorithm. It states that the usefulness of available physical resource elements (located, e.g., on an extraterrestrial planet) remains unchanged in the absence of artificial self-reproducing entities.

Using the relaxed notion of weak regularity, we now make an assumption about the structure of the given generation system that is less restrictive than similar assumptions made by all other seeding approaches. In nature, it is not possible for a progenitor to cannibalize an offspring to produce that same offspring, and it would be impractical to require this capacity in an artificial self-reproducing system. However, cannibalization of offspring to produce other offspring has, in general, been documented, which contradicts an assumption of strong regularity in previous seeding approaches. We also note that families, as defined above, are disjoint in nature.

Assumption 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.

Finally, we assume knowledge of the cost of the machines and resources in the self-reproducing system. For instance, in situ resources may be less expensive than resources that need to be launched with the seed machines of the lunar robotic colony.

Assumption 6. The function $J : M \to \mathbb{R}$ representing the cost of machines, and the function $K : R \to \mathbb{R}$ representing the cost of resource ordered lists, are both provided. Let $K(r\backslash x) \leq K(r)$.

The functions J and K are time-independent; indeed, the amount of time that is required for self-reproduction is not accounted for by the method in this article. We leave the recognition and compensation for time constraints during seeding, which may correspondingly affect seed cost as well as the individual costs of machines and resource ordered lists, to future endeavors.

3.3 Seed Identification Problem

The problem tackled in this article is the minimization, under Assumptions 1 through 6, of the total cost of the machines and resource ordered lists in a seed when the given generation function is used. We explain this problem with the example directed-graph representation of a family in Figure 6.

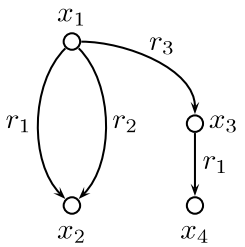


Figure 6. Directed-graph representation of a family to illustrate that seed cost cannot be minimized solely by examining the cost of the elements of a seed set. If r_2 costs less than r_1, then a seed resource set of $\{r_1, r_2, r_3\}$ may be less expensive than a seed resource set of $\{r_1, r_3\}$, because a portion of the more costly r_1 in the latter set can be replaced by r_2.

In Figure 6, let the cost of the resource ordered list r_2 be less than the cost of the resource ordered list r_1. Intuitively, there are two seeds for this example:

(1) $\{x_1\} \cup \{r_1, r_3\}$, and

(2) $\{x_1\} \cup \{r_1, r_2, r_3\}$.

Although the first seed has a resource seed set with lower cardinality than the second seed, the first seed is nonoptimal. This is because r_1, which is more expensive than r_2, must be utilized by x_1 to generate x_2 with the first seed. It is less costly if x_1 utilizes r_2 to produce x_2. The resource ordered list r_1 is required to generate x_4 in both seeds; however, a less expensive amount of r_1 and r_2 is required in the second seed than the total amount of r_1 in the first seed. Therefore, optimality of the seed set cannot be determined solely by examining the cost of the elements of the set. The effect of the generation function must be considered as well.

More specifically, if there exist two sequences of resource ordered lists, (r_m) and (r_n), where each sequence possesses a common resource $r \in R$, and if $y = G(x, (r_m))$ and $z = G(x, (r_n))$ in the given generation system to be seeded, then a solution to the seed identification problem will charge the cost of r twice.

3.4 Approach to Seed Identification

The idea behind the SI algorithm is that 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 subfamily.

Definition 10. The *generation subsystem of machine* x_1 is the generation system $\Gamma_{x_1} = (U, M_{x_1}, R_{x_1}, G|_{M_{x_1} \times R_{x_1}})$ where

$$M_{x_1} = \bigcup_{i=0}^{\infty} M_{x_1}^i, \tag{6}$$

$$M_{x_1}^i = x \in M | \ \exists (r_i) \text{ from } R : x = G(x_1, (r_i)), \tag{7}$$

$$R_{x_1} = \bigcup_{i=0}^{\infty} R_{x_1}^i, \tag{8}$$

$$R_{x_1}^i = \bigcup_{|R|^i} (r_i) \text{ from } R \mid G(x_1, (r_i)) \in M. \tag{9}$$

In Definition 10, $M_{x_1}^i$ is the set of all the descendants of x_1 produced after i generations, M_{x_1} is the set of all the descendants of x_1, $R_{x_1}^i$ is the set of all resource-ordered-list sequences of length i that would produce a descendant of x_1, and R_{x_1} is the set of all resource-ordered-list sequences that would produce a descendant of x_1. Hence, the generation subsystem of x_1 is the largest family for which x_1 is a matriarch. Although M and R are finite, the infinite unions refer to the possibly infinite number of descendants produced by a matriarch.

We 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. Let M_{φ} denote the set of matriarchs.

In the generation subsystem of a matriarch x_{φ}, every machine in the subsystem can be produced except possibly x_{φ} itself. Thus, in the course of seeding the subsystem of x_{φ}, the machines to pick for the seed set of the subsystem, $S_{x_{\varphi}}$, are x_{φ} and certain machines contained in $R_{x_{\varphi}}$.

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 in [32].

Theorem 4. *Assume that the generation subsystem of the machine* x_1, $\Gamma_{x_1} = (U, M_{x_1}, R_{x_1}, G|_{M_{x_1} \times R_{x_1}})$, *is weakly regular. Consider a single lineage in* Γ_{x_1}. *Let* $y \in M_{x_1}$, *and* (r_{m+1}) *be a sequence of resource ordered lists such that* $G(x_1, (r_{m+1})) = y$, *that is,* y *belongs to the lineage of* x_1 *through* (r_{m+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) $\exists r$ *constituting* $(r_m) : y \prec r$.

(4) m *is the smallest natural number for which assumptions (1)–(3) hold.*

Then a seed set for the weakly regular subfamily $(U, X, Z, G|_{X \times Z})$, *where* X *is the set of machines of* (x_{m+1}) *and* Z *is the set of resource ordered lists of* (r_m), *is* $S = \{x_1, y\} \cup Z \backslash M_{x_1}$, *and* S *has minimum* $|M_S|$.

Proof. See Appendix. □

Theorem 4 states that if a sequence of m resource ordered lists, (r_m), is used to produce a sequence of $m + 1$ machines, (x_{m+1}), and the machine y is contained in a resource ordered list belonging to the sequence (r_m) but does not itself belong to the sequence (x_{m+1}), and the resource seed set is devoid of machines, then the machine seed set must consist of y and the first machine in (x_{m+1}). We call y an *irregular* machine.

For instance, consider the weakly regular generation system of Figure 4a and Figure 5b. Let $m = 1$ and $y = x_3$. Since the assumptions of Theorem 4 hold, a seed set for the generation system is $S = \{x_1, x_3\} \cup \{r_1 \backslash x_3, r_2\}$, which is rather intuitive.

As a result of Theorem 4, we can examine the sequences of machines generated by a matriarch when seeding weakly regular generation systems. We first present a *lineage seeding* (LS) subalgorithm, before giving the general SI algorithm.

3.5 The LS Subalgorithm

Algorithm 1. Lineage seeding.

Input: a lineage of a matriarch, x_1, through (r_n).

Output: a seed set $S = M_S \cup R_S$ for this lineage, where

$M_S = x_1 \cup \{\text{all irregular machines}\}$,

$R_S = \{(r_n)\} \backslash M_{x_1}$.

1: $M_S \leftarrow \{x_1\}$.

2: initialize a linked list, L, with one element, x_1.

3: **for** $1 \leq i \leq n$ **do**

4: let $y \leftarrow G(x_1, (r_i))$.

5: **if** y is not in L **then**

6: **for** each machine in $\{(r_i)\} \backslash \{x_1, y\}$ that is not in L **do**

7: add the machine to the tail of L.

8: **end for**

9: add y to the tail of L.

10: **else**

11: insert machines that are contained in $\{(r_i)\}\backslash\{x_1, y\}$ that are not already in L into list positions that immediately precede y.

12: **if** $G(x_1, (r_j))$, $1 \leq j < i$, and all contained machines in $\{(r_i)\}\backslash\{x_1, y\}$ have positions in L that are not between the positions of x_1 and y **then**

13: y is an irregular machine. $M_S \leftarrow M_S \cup \{y\}$.

14: **end if**

15: **end if**

16: **end for**

17: $R_S \leftarrow \{(r_n)\}\backslash M_{x_1}$.

18: $S \leftarrow M_S \cup R_S$.

At each iteration of Algorithm 1, the LS subalgorithm, the number of machines of the lineage that have been examined grows by one. If the lineage at that iteration is not strongly regular, then Theorem 4 comes into play. The linked list L that is utilized by the LS subalgorithm is a tool to indicate which machines should be added to the machine seed set.

Theorem 5. *The LS subalgorithm is correct. That is, the output of the LS subalgorithm is a seed for a lineage of a matriarch, x_1, through (r_n).*

Proof. See Appendix. □

3.6 The SI Algorithm

In lines 1 through 3 of Algorithm 2, the SI algorithm, each machine (vertex) is a starting point (root) in the initial generation system (directed graph), and we need to find the generation subsystem of that machine (maximally connected subgraph that can be 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 [2, 12, 25].

In lines 4 and 21 of Algorithm 2, the idea is to seed a primary generation subsystem first, and then go back to a secondary generation subsystem $M\backslash M_{x_j}$ and partition and seed iteratively.

In lines 5 through 19 of Algorithm 2, we ensure that the primary generation subsystem has the property that each offspring is generated from only one resource ordered list. Thereafter, we can select all resource ordered lists to be a part of the seed set. We use the Chu-Liu-Edmonds algorithm [15, 44] to find a directed minimum spanning tree (DMST) for each matriarch in the primary generation subsystem (i.e., we find a family tree). 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 subalgorithm 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 seed with minimum cost.

Algorithm 2. Seed identification.

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

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

1: **for all** $x_i \in M$, $1 \leq i \leq n$ **do**
2: determine Γ_{x_i}.
3: **end for**
4: select the Γ_{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.
5: **for all** the matriarchs of the largest generation subsystem **do**
6: **if** in the graph representation $\mathcal{G}_{x_i}$ of Γ_{x_i}, x_i has entering edges **then**
7: add a new vertex x_i'.
8: change these entering edges so that they now enter x_i'.
9: **end if**
10: label each edge r of $\mathcal{G}_{x_i}$ with the cost of the resource represented by that edge, $K(r)$. For each edge r that exits a vertex y, add the cost of the machine represented by that vertex, $J(y)$, to the cost $K(r)$.
11: find a directed minimum spanning tree (DMST) in $\mathcal{G}_{x_i}$, the graph of Γ_{x_i} with root at x_i.
12: $\mathcal{G}_{x_i min} \leftarrow$ this DMST of Γ_{x_i}.
13: **for all** simple paths in $\mathcal{G}_{x_i \min}$ **do**
14: use the LS subalgorithm to seed each path and obtain S_{path_j}.
15: **end for**
16: $S_{x_i} \leftarrow \bigcup_j S_{path_j}$. Let S_{x_i} be the union of a machine set, $M_{S_{x_i}}$, and a set of resource ordered lists, $R_{S_{x_i}}$.
17: $\mathcal{J}_{x_i} \leftarrow$ the cost of $\mathcal{G}_{x_i \min}$ − the cost of nonirregular machines removed by the LS subalgorithm.
18: **end for**
19: select the S_{x_i} for which $\mathcal{J}_{x_i}$ is a minimum.
20: add x_i to the set of matriarchs, $M_{\text{♀}}$.
21: remove all $x \in M_{x_i}$ from M.
22: **if** $M \neq \emptyset$ **then**
23: go to Line 4.
24: **else**
25: $S \leftarrow \bigcup_{x_i \in M\text{♀}} S_{x_i}$.
26: **end if**

3.7 Properties of the SI Algorithm

Theorem 6. *The SI algorithm is correct. That is, the output of the algorithm is an optimal seed for the given generation system.*

Proof. See Appendix. □

Remark 3. The assumption of disjoint families in Assumption 5 is one requisite for optimality of the seed using the SI algorithm. Although the proposed algorithm will work if the given generation system has

families that are not disjoint, the resultant seed may or may not be optimal (i.e., no claims about optimality can be asserted). However, we conjecture that the resultant seed will be "close" to optimal. Hence our statement at the start of Section 3 about partially handling difficulty (a) in Section 2.2. If the given generation system does not possess disjoint families, then line 20 of the SI algorithm should read "go to line 1."

Theorem 7. *The SI algorithm is complete. That is, the algorithm will output a seed if one exists for the given generation system.*

Proof. See Appendix. □

Theorem 8. *The SI algorithm is guaranteed to stop after a finite number of iterations. The time complexity for the operation of this algorithm is polynomial in* $|M|$ *and* $|\mathrm{R}|$.

Proof. See Appendix. □

4 Example Applications of the SI Algorithm

Two examples are provided in this section. The first example serves to illustrate the broad workings of the SI algorithm by rendering the LS subalgorithm trivial. The second example illustrates the details of the LS subalgorithm and its relationship with the SI algorithm.

4.1 A Strongly Regular Self-Replicating System

We can use generation theory and the algorithm in this article to analyze a modified version of the *semi-autonomous replicating system* designed by Chirikjian et al. [7, 43]. This self-replicating LEGO Mindstorms system was one of the first working models of a multifunctional self-reproducing robot that could constitute a lunar robotic colony. The semi-autonomous replication process of the Suthakorn-Kwon-Chirikjian robot required the progenitor robot to commute along painted lines between stations to maneuver and assemble LEGO Mindstorms kit components together (see [7] for illustrations). Each station facilitated a replica robot assembly task, for example, controller-chassis assembly, motor and track assembly, or gripper assembly.

In the original design, we can take M to be the set of all entities that are each made up of two or more LEGO Mindstorms kit components fixed together in some way. Let

$$M := \{x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9\} \tag{10}$$

and

$$R := \{r_1, r_2, r_3, r_4, r_5, r_6, r_7, r_8, r_9\}, \tag{11}$$

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

$x_1 :=$ 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 (RCX); 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],

$x_1 = G(x_1, r_9)$.

The original Suthakorn-Kwon-Chirikjian generation system is a single family, and so the SI algorithm terminates after one iteration. If we take into account the necessity of batteries for operation, the application becomes nontrivial. 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 that can charge uncharged batteries. Thus, the following changes to the system need to be made:

$$M := \{x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}\}, \tag{12}$$

$$R := \{r_1, r_2, r_3, r_4, r_5, r_6, r_7, r_8, r_9, r_{10}\}, \tag{13}$$

x_{10} := battery charger,

r_{10} := [electricity; uncharged batteries],

x_{11} := charged batteries,

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

$r_3 :=$ [robot control system; x_3; x_{11}].

It follows that the directed-graph representation of this modified generation system is as indicated in Figure 7.

This generation system is strongly regular, and is made up of two disjoint families corresponding to the two disjoint subgraphs of Figure 7. Applying the SI 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. For this example, we let $\forall y \in M$, $J(y) := 1$, and $\forall r \in R$, $K(r) :=$ the number of elements in the ordered list of r.

The machine sets of Γ_{x_i}, $1 \leq i \leq 11$, are the following:

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

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

$M_{x_3} = \emptyset$,

$M_{x_4} = \emptyset$,

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

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

$M_{x_7} = \emptyset$,

$M_{x_8} = \emptyset$,

$M_{x_9} = \emptyset$,

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

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

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.

Consider x_1. Since x_1 has entering edges in Figure 7, we define a new vertex x_1' and change these edges so that they now enter x_1'. The DMST with root at x_1 yields $R_{x_1 \min} = \{r_1, r_2, r_3, r_4, r_5, r_6, r_7, r_8, r_9\}$.

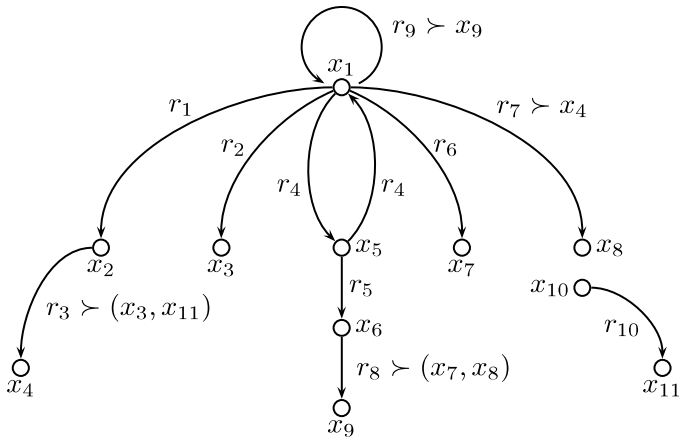


Figure 7. Directed-graph representation of the modified Suthakorn-Kwon-Chirikjian semi-autonomous replicating system, which is a strongly regular generation system. With the provided cost functions, the SI algorithm yields the intuitive seed $S = \{x_1, x_{10}\} \cup \{r_1, r_2, r_3\backslash[x_3; x_{11}], r_4, r_5, r_6, r_7\backslash x_4, r_8\backslash[x_7; x_8], r_9\backslash x_9, r_{10}\}$.

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. The LS subalgorithm applied to each simple path returns $S_{x_1} = \{x_1\} \cup R_{x_1\min} \backslash M$. We obtain $\mathcal{J}_{x_1} = 35 - 6 = 29$.

Similarly, for x_5, we define a new vertex x_5' and change the edges that enter x_5 so that they now enter x_5'. The DMST with root at x_5 yields $R_{x_5\min} = \{r_1, r_2, r_3, r_4, r_5, r_6, r_7, r_8\}$. The DMST algorithm does not select r_9. The LS subalgorithm applied to each simple path returns $S_{x_5} = \{x_5\} \cup R_{x_5\min} \backslash M$. We obtain $\mathcal{J}_{x_5} = 38 - 5 = 33$.

Of the two matriarchs, S_{x_1} is selected, since it has lower cost. Thus, the seed for the first family is

$$S_{x_1} = \{x_1\} \cup \{r_1, r_2, r_3 \backslash [x_3; x_{11}], r_4, r_5, r_6, r_7 \backslash x_4, r_8 \backslash [x_7; x_8], r_9 \backslash x_9\}. \tag{14}$$

Continuing the SI algorithm, we remove the machines that are in the generation subsystem of x_5, leaving us with $M = \{x_{10}, x_{11}\}$.

We now iterate through the algorithm, selecting x_{10}, since it has the larger generation subsystem. The DMST yields $R_{x_{10}\min} = \{r_{10}\}$, and the LS subalgorithm gives us one possible seed. Thus, the seed for the second family is

$$S_{x_{10}} = \{x_{10}\} \cup \{r_{10}\}. \tag{15}$$

Removing the machines in the generation subsystem of x_{10} leaves us with $M = \emptyset$. Therefore, a seed for the self-replicating system is

$$S = \{x_1, x_{10}\} \cup \{r_1, r_2, r_3 \backslash [x_3; x_{11}], r_4, r_5, r_6, r_7 \backslash x_4, r_8 \backslash [x_7; x_8], r_9 \backslash x_9, r_{10}\}. \tag{16}$$

We have thus arrived at a very intuitive result—the prototype robot and the battery charger can initiate the semi-autonomous replicating system. Although the example is simple and confirms intuition, the SI algorithm procedure is clearly illustrated as a consequence.

4.2 A Weakly Regular Self-Replicating System

The example in this section serves to illustrate the working of lines 5 through 19 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 [24]. It is worth noting that intricate generation systems similar to this example appear readily in nature, yet such a level of sophistication has not been observed in human-designed self-reproducing systems so far.

The directed-graph representation of the naturally occurring atmospheric ozone cycle attacked by chlorine is drawn in Figure 8, using the generation system model

$$M := \{x_1, x_2, x_3, x_4, x_5\}, \tag{17}$$

$$R := \{r_1, r_2, r_3, r_4, r_5\}, \tag{18}$$

where we define each of the constituent machines, resource ordered lists, and generation steps as follows:

$x_1 :=$ 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],

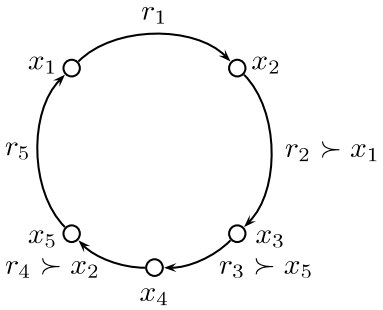


Figure 8. Directed-graph representation of the atmospheric ozone cycle attacked by chlorine, which is a weakly regular generation system. With the provided cost functions, the SI algorithm suggests the seed $S = \{x_5\} \cup \{r_1, r_2 \backslash x_1\}$.

$x_3 := O_3$, or ozone molecules,

$x_3 = G(x_2, r_2)$,

$x_4 := ClO + O_2$,

$x_5 := Cl + O_2$,

$r_3 := [x_5]$ (although note that only Cl is required),

$x_4 = G(x_3, r_3)$,

$r_4 := [x_2]$,

$x_5 = G(x_4, r_4)$,

$r_5 := [\ ]$, an empty resource ordered list to remove Cl from x_5,

$x_1 = G(x_5, r_5)$.

This self-replicating system is made up of a single family, as indicated by the connected graph of Figure 8. The family is not strongly regular; for instance, starting with the machine x_3, we would need x_5 before we had produced it. However, the family is weakly regular; no offspring machine is contained in a resource ordered list used to generate that machine. We let $\forall y \in M, J(y) := 1$, and $\forall r \in R$, $K(r) :=$ the number of elements in the ordered list of r.

Every machine in this generation cycle is a matriarch for the family, and so the LS subalgorithm has to produce a linked list for the lineage of each machine. If we start with x_1, then the LS subalgorithm yields the following list and machine seed set:

$$L_{x_1} = [x_1; x_2; x_3; x_5; x_4], \tag{19}$$

$$M_{S_{x_1}} = \{x_1, x_5\}, \tag{20}$$

$$\mathcal{J}_{x_1} = 10 - 2 = 8. \tag{21}$$

Similarly, starting with x_2 yields

$$L_{x_2} = [x_2; x_1; x_3; x_5; x_4], \tag{22}$$

$$M_{S_{x_2}} = \{x_2, x_5, x_1\}, \tag{23}$$

$$\mathcal{J}_{x_2} = 10 - 1 = 9. \tag{24}$$

Starting with x_3 yields

$$L_{x_3} = [x_3; x_2; x_5; x_4; x_1], \tag{25}$$

$$M_{S_{x_3}} = \{x_3, x_5, x_2\}, \tag{26}$$

$$\mathcal{J}_{x_3} = 10 - 1 = 9. \tag{27}$$

Starting with x_4 yields

$$L_{x_4} = [x_4; x_2; x_5; x_1; x_3], \tag{28}$$

$$M_{S_{x_4}} = \{x_4, x_2\}, \tag{29}$$

$$\mathcal{J}_{x_4} = 10 - 2 = 8. \tag{30}$$

Finally, starting with x_5 yields

$$L_{x_5} = [x_5; x_1; x_2; x_3; x_4], \tag{31}$$

$$M_{S_{x_5}} = \{x_5\}, \tag{32}$$

$$\mathcal{J}_{x_5} = 10 - 3 = 7. \tag{33}$$

Hence, for the cycle in Figure 8, we select the seed where

$$M_S = \{x_5\} \tag{34}$$

and

$$R_S = \{r_1, r_2 \backslash x_1\}. \tag{35}$$

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.

5 Conclusions and Future Work

A novel algorithm to identify an optimal seed for a general class of generation systems has been proposed. It utilizes the concepts of families and weak regularity to consider resource ordered lists and their composition, deal with machines of deficient rank that are used in resource ordered lists, isolate seed machines from generation cycles, and overcome the difficulty of seeding self-reproducing systems where the generation of a copy 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. Modifying the algorithm in this article to ensure that a seed is robust to the probabilistic selection of resources (i.e., the system is still capable of self-replication and self-reproduction) is another possible extension of this work. With the theory in place to analyze generation systems, the next step is to synthesize generation systems.

The SI algorithm needs to be extended to (1) allow for the determination, whenever possible, of a seed of prespecified order $\nu\mu$; (2) incorporate some notion of the quantity of seed resources needed to perpetuate a system; and (3) recognize and compensate for time constraints that may impose a larger-size seed upon the system.

Acknowledgments

We are very grateful to the anonymous reviewers for their valuable comments and suggestions that have improved the quality of this work.

References

1. Adey, W. H., & Loveland, K. (1998). *Dynamic aquaria* (2nd ed.). San Diego: Academic Press.
2. Aho, A. V., Hopcraft, J. E., & Ullman, J. D. (1974). *The design and analysis of computer algorithms*. Reading, MA: Addison-Wesley.
3. Butler, Z., Fitch, R., & Rus, D. (2002). Distributed control for unit-compressible robots: Goal recognition, locomotion, and splitting. *IEEE/ASME Transactions on Mechatronics*, *7*(4), 418–430.
4. Butler, Z., Kotay, K., Rus, D., & Tomita, K. (2002). Cellular automata for decentralized control of self-reconfigurable robots. In *Proceedings of the 2002 IEEE International Conference on Robotics and Automation*, vol. 1 (pp. 809–816).
5. Castano, A., Behar, A., & Will, P. M. (2002). The Conro modules for reconfigurable robots. *IEEE/ASME Transactions on Mechatronics*, *7*(4), 403–409.
6. Chirikjian, G. S. (2008). Parts entropy, symmetry, and the difficulty of self-replication. In *Proceedings of the 2008 ASME Dynamic Systems and Control Conference*, DSCC2008-2280.
7. Chirikjian, G. S., Zhou, Y., & Suthakorn, J. (2002). Self-replicating robots for lunar development. *IEEE/ASME Transactions on Mechatronics*, *7*(4).
8. Christensen, A. L., O'Grady, R., & Dorigo, M. (2007). Morphology control in a multirobot system: Distributed growth of specific structures using directional self-assembly. *IEEE Robotics & Automation Magazine*, *14*(4), 18–25.
9. Christensen, D. J. (2007). Experiments on fault-tolerant self-reconfiguration and emergent self-repair. In *Proceedings of the IEEE Symposium on Artificial Life* (pp. 355–361).
10. Christensen, D. J. (2008). *Elements of autonomous self-reconfigurable robots*. Ph.D. thesis, University of Southern Denmark.
11. Chu, Y. J., & Liu, T. H. (1965). On the shortest arborescence of a directed graph. *Scientia Sinica*, *14*, 1396–1400.
12. Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2001). *Introduction to algorithms* (2nd ed.). Cambridge, MA: MIT Press.
13. Diestel, R. (2005). *Graph theory* (3rd ed.). New York: Springer.
14. Edmonds, J. (1967). Optimum branchings. *Journal of Research of the National Bureau of Standards*, *71B*(4), 233–240.
15. Even, S. (1979). *Graph algorithms*. Potomac, MD: Computer Science Press.
16. Fitch, R., & Rus, D. (2003). Self-reconfiguring robots in the USA. *Journal of the Robotics Society of Japan*, *21*(8), 4–10.
17. Foing, B. (2005). Roadmap for robotic and human exploration of the moon and beyond. In *Proceedings of the 56th International Astronautical Congress*, IAC-05-A5.1.01.
18. Freitas, R. A., Jr., & Gilbreath, W. P. (Eds.) (1982). *Advanced automation for space missions*. N83-15348 (NASA Conference Publication 2255). NASA.
19. Freitas, R. A., Jr., & Merkle, R. C. (2004). *Kinematic self-replicating machines*. Georgetown, TX: Landes Bioscience.

20. Fukuda, T., Nakagawa, S., Kawauchi, Y., & Buss, M. (1988). Self-organizing robots based on cell structures—CEBOT. In *Proceedings of the 1988 IEEE/RSJ International Conference on Intelligent Robots and Systems* (pp. 145–150).

21. Gilpin, K., Kotay, K., & Rus, D. (2007). Miche: Modular shape formation by self-disassembly. In *Proceedings of the 2007 IEEE International Conference on Robotics and Automation* (pp. 2241–2247).

22. Goldstein, S. C., Campbell, J. D., & Mowry, T. C. (2005). Programmable matter. *IEEE Computer*, *38*(6), 99–101.

23. Griffith, S., Goldwater, D., & Jacobson, J. M. (2005). Self-replication from random parts. *Nature*, *437*, 636.

24. Hobbs, P. V. (2000). *Introduction to atmospheric chemistry*. Cambridge, UK: Cambridge University Press.

25. Hopcroft, J., & Tarjan, R. (1973). Algorithm 447: Efficient algorithms for graph manipulation. *Communications of the ACM*, *16*(6), 372–378.

26. Howe, A. S. (2007). Self-assembling modular robotic structures: Standard units with swappable payloads. *IEEE Robotics & Automation Magazine*, *14*(4), 26–33.

27. Jørgensen, M. W., Østergaard, E. H., & Lund, H. H. (2004). Modular ATRON: Modules for a self-reconfigurable robot. In *Proceedings of the 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 2 (pp. 2068–2073).

28. Kabamba, P. T., Owens, P. D., & Ulsoy, A. G. (2011). The von Neumann threshold of self-reproducing systems: Theory and application. *Robotica*, *29*(1), 123–135.

29. Klavins, E. (2007). Programmable self-assembly. *IEEE Control Systems Magazine*, *27*(4), 43–56.

30. Lodish, H., Berk, A., Matsudaira, P., Kaiser, C. A., Krieger, M., Scott, M. P., Zipursky, L., & Darnell, J. (2004). *Molecular cell biology* (5th ed.). New York: W. H. Freeman.

31. Menezes, A., & Kabamba, P. (2007). A combined seed-identification and generation analysis algorithm for self-reproducing systems. In *Proceedings of the 2007 American Control Conference* (pp. 2582–2587).

32. Menezes, A., & Kabamba, P. (2007). An optimal-seed identification algorithm for self-reproducing systems. In *Proceedings of the 58th International Astronautical Congress*, IAC-07-D3.2.02.

33. Menezes, A. A. (2010). *Selective evolutionary generation systems: Theory and applications*. Ph.D. thesis, University of Michigan.

34. Menezes, A. A., & Kabamba, P. T. (2008). Resilient self-reproducing systems. In *Proceedings of the 2008 ASME Dynamic Systems and Control Conference*.

35. Murata, S., & Kurokawa, H. (2007). Self-reconfigurable robots: Shape-changing cellular robots can exceed conventional robot flexibility. *IEEE Robotics & Automation Magazine*, *14*(1), 71–78.

36. Murata, S., Yoshida, E., Kamimura, A., Kurokawa, H., Tomita, K., & Kokaji, S. (2002). M-TRAN: Self-reconfigurable modular robotic system. *IEEE/ASME Transactions on Mechatronics*, *7*(4), 431–441.

37. Østergaard, E. H. (2004). *Distributed control of the ATRON self-reconfigurable robot*. Ph.D. thesis, University of Southern Denmark.

38. Salemi, B., Moll, M., & Shen, W.-M. (2006). SUPERBOT: A deployable, multi-functional, and modular self-reconfigurable robotic system. In *Proceedings of the 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems* (pp. 3636–3641).

39. Shen, W.-M., Will, P., Galstyan, A., & Chuong, C.-M. (2004). Hormone-inspired self-organization and distributed control of robotic swarms. *Autonomous Robots*, *17*(1), 93–105.

40. Sipper, M. (1998). Fifty years of research on self-replication: An overview. *Artificial Life*, *4*(3), 237–257.

41. Stoy, K., Brandt, D., & Christensen, D. J. (2010). *Self-reconfigurable robots: An introduction*. Cambridge, MA: MIT Press.

42. Stoy, K., & Nagpal, R. (2004). Self-repair through scale-independent self-reconfiguration. In *Proceedings of the 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 2 (pp. 2062–2067).

43. Suthakorn, J., Kwon, Y. T., & Chirikjian, G. S. (2003). A semi-autonomous replicating robotic system. In *Proceedings of the 2003 IEEE International Symposium on Computational Intelligence in Robotics and Automation*.

44. Tarjan, R. E. (1977). Finding optimum branchings. *Networks*, *7*, 25–35.

45. Ünsal, C., Kiliççöte, H., & Khosla, P. K. (2001). A modular self-reconfigurable bipartite robotic system: Implementation and motion planning. *Autonomous Robots*, *10*(1), 23–40.

46. Varshavskaya, P., Kaelbling, L. P., & Rus, D. (2004). Learning distributed control for modular robots. In *Proceedings of the 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 3 (pp. 2648–2653).

47. von Neumann, J. (1966). *Theory of self-reproducing automata*. Urbana, IL: University of Illinois Press.

48. Walter, J. E., Welch, J. L., & Amato, N. M. (2004). Distributed reconfiguration of metamorphic robot chains. *Distributed Computing*, *17*(2), 171–189.

49. Wertz, J. R., & Larson, W. J. (Eds.) (1999). *Space mission analysis and design* (3rd ed.). El Segundo, CA: Microcosm Press.

50. Yim, M., Roufas, K., Duff, D., Zhang, Y., Eldershaw, C., & Homans, S. (2003). Modular reconfigurable robots in space applications. *Autonomous Robots*, *14*(2–3), 225–237.

51. Yim, M., Shen, W.-M., Salemi, B., Rus, D., Moll, M., Lipson, H., Klavins, E., & Chrikjian, G. S. (2007). Modular self-reconfigurable robot systems: Challenges and opportunities for the future. *IEEE Robotics & Automation Magazine*, *14*(1), 43–52.

52. Yim, M., Shirmohammadi, B., Sastra, J., Park, M., Dugan, M., & Taylor, C. J. (2007). Towards robotic self-reassembly after explosion. In *Proceedings of the 2007 IEEE/RSJ International Conference on Intelligent Robots and Systems* (pp. 2767–2772).

53. Zykov, V., Mytilinaios, E., Adams, B., & Lipson, H. (2005). Self-reproducing machines. *Nature*, *435*, 163–164.

Appendix: Proofs of Theorems

Theorem 1.

Proof. We start with the definition of a strongly regular generation system: Whenever $y = G(x, (r_\mu))$, where x and y are machines and (r_μ) is a sequence of μ resource ordered lists, we have $y \not\prec r$ for all ordered lists r that constitute the sequence (r_μ). Let x' denote the machine that produces the machine y using the resource ordered list r_μ, that is, $y = G(x', r_\mu)$. Since $y \not\prec r$ for all r constituting (r_μ), we have $y \not\prec r_\mu$. Thus, the definition of a weakly regular generation system is satisfied. □

Theorem 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 $(x, y) \in M \times M$, $\exists z \in M$, and (r_n) and (r_m) from 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 constituting (r_n), and a path from z to y through the sequence of edges constituting (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. □

Theorem 3.

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 4.

Proof. This proof follows directly from the definition of a seed. First, we are given that there exists an r constituting (r_m) such that $y \prec r$. Since $\forall 1 \leq i \leq m$, $x_{i+1} = G(x_1, (r_i))$, a seed for the weakly regular subfamily $(U, X, Z, G|_{X \times Z})$, where X is the set of machines of (x_{m+1}) and Z is the set of resource ordered lists of (r_m), is

$$\begin{aligned} S &= \{x_1\} \cup Z \\ &= \{x_1, y\} \cup Z \backslash \{y\}. \end{aligned}$$

Since m is the smallest natural number for which assumptions (1)–(3) hold, m is the first location at which weak regularity occurs in the single lineage. Hence, the lineage is strongly regular at positions prior to m. It follows that we can replace $Z\backslash\{y\}$ with $Z\backslash M_{x_1}$ so that the resource seed set is devoid of machines, and this replacement does not affect the validity of S as a seed.

If $R_S \cap M_S = \emptyset$, then we need to have $|M_S| \geq 1$ by Definition 6, 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, (r_m) has a resource that contains y, and y is in the lineage of x_1 through (r_m), but y cannot be generated by x_1. Therefore, the system needs to be started with both x_1 and y. Thus, $|M_S| = 2$, the minimum possible. □

Theorem 5.

Proof. This proof uses mathematical induction. We assume that we are given a lineage of a matriarch, x_1, through (r_n). Let M_{x_1} be the set of all descendants of x_1, and $\{(r_n)\}$ be the set of resource ordered lists in the lineage of x_1. Let $R_S = \{(r_n)\}\backslash M_{x_1}$, and $M_S = \{x_1\}$. Let r be the first resource ordered list in this lineage, and s be the second. Let $y := G(G(x_1, r), s)$, different from $G(x_1, r)$.

Consider $G(x_1, r)$. If $y \prec r$, the subalgorithm takes $M_S \leftarrow M_S \cup \{y\}$, and by Theorem 4, the new M_S forms a seed for the path when unioned with R_S. Otherwise, the original M_S is still a seed when unioned with R_S, because 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 resource ordered lists, (r_{k-1}). Let $y := G(G(x_1, (r_{k-1})), r_k)$, different from $G(x_1, (r_{k-1}))$.

Consider $G(x_1, (r_{k-1}))$. If y is contained in a resource ordered list of (r_{k-1}), the subalgorithm takes $M_S \leftarrow M_S \cup \{y\}$, and by Theorem 4, the new M_S forms a seed for the path when unioned with R_S. Otherwise, the original M_S is still a seed when unioned with R_S, because y is not required. □

Theorem 6.

Proof. We have to first prove that the output set S is a seed for the initial self-reproducing system. Since Γ is a union of families, and $S = \cup S_x$ for x belonging to the set of matriarchs M_φ, it suffices to prove that each S_x is a seed for one of the constituent families. Thus, we will show that each of the intermediate steps is correct.

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 and the DFS algorithm are able to correctly find the vertices reachable from a root in a weakly connected directed graph [12]. Thus, the use of either of these algorithms ensures that this step is correct.

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.

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 [11, 14, 44]. We have shown by Theorem 5 that the LS subalgorithm is correct for any path in the tree. Since the union of seed sets is itself a seed set, $S_x = \cup S_{\text{path}}$ is a valid seed. Just as in the previous step, 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.

We now demonstrate optimality.

Let $\Gamma = (U, M, R, G)$ be made up of k weakly regular disjoint families. 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 resource ordered lists 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. To allow for the cost of a machine that utilizes a resource ordered list, this cost is included in the resource ordered list cost in the directed-graph representation. The DMST that is obtained via the Chu-Liu-Edmonds algorithm finds a path between the root vertex and all other vertices with minimal cost. We have shown that each pass through the SI algorithm produces a seed for a family, before the family is removed from the original generation system. The seed set chosen for the family is the set with the lowest total cost of machines and resource ordered lists, and is selected from all the possible matriarch seed sets for that 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. 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 for the original generation system that is optimal with respect to cost. □

Theorem 7.

Proof. We have to show that if a seed exists, the algorithm in this article will output one possible seed. Consider that a seed for a generation system always exists—namely, the trivial seed, consisting of all the machines and resource ordered lists in the generation system, that is, $S = M \cup R$. Indeed, the algorithm presumes this seed at the start, before removing redundant resource ordered lists and machines that belong to a matriarch's subsystem. Theorem 6 shows that the output of the algorithm is a seed.

Thus, completeness is guaranteed. □

Theorem 8.

Proof. The LS subalgorithm is convergent because no cycles 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.

We now consider the time complexity of operation during the first iteration of the algorithm. Let $|M| = n$ and $|R| = m$.

The use of either one of the BFS or DFS algorithms has time complexity $O(n + m)$ [12].

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 $O(n)$.

The time complexity of the DMST algorithm is $O(n_p m_p)$ [15], where n_p is the number of machines in the primary subsystem, and m_p is the number of resource ordered lists 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 subalgorithm 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 resource ordered lists have to be visited to remove any contained machines results in a time complexity of $O(m_p)$. Allowing 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.

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 during the first iteration of the algorithm is of polynomial order in n and m. □