Да, возможны более осмысленные и в принципе более «точные» подходы, чем слепой/эвристический генетический поиск по пространству 1D-правил. Но это потребует смены угла зрения:

- от «перебора матриц переходов» к
- **синтезу правил как решений структурной/топологической задачи**
  (типа: «найти такие локальные преобразования, чтобы фундаментальная группа, спектр Ω-циклов и coarse-поля имели заданный вид»).

Ниже дам:

1. Чёткую формулировку задачи в терминах групп и топологии.
2. Какие именно инварианты можно требовать: Ω-циклы, π_1, законы сохранения, спектр длин циклов.
3. План конструктивного синтеза правил через:
   - групповые представления,
   - ограничения на Cayley-граф локальной динамики,
   - и уравнения на матрицу переходов.
4. Как это связать с тем, что у вас уже есть в `sm_evolutionary_search.pdf` и observer-онтологии.
5. Где граница «точно/конструктивно» и где всё равно неизбежны эвристики.

1. Что вы уже делаете и где здесь «слепой» компонент

Сейчас у вас:

- Геном RuleGenome: матрица переходов 2^L × 2^L, L=3:

  ```
  transitions[i,j] = 1 означает правило: pattern_i → pattern_j
  ```

- Ограничения:
  - обратимость: если transitions[i,j]=1, то transitions[j,i]=1;
  - максимум один выход из паттерна (частичная функция).
- Эволюционный поиск:

  ```
  population_size ≈ 50
  generations ≈ 50
  pattern_length = 3
  ```

- fitness = SMFitness(lattice_size=150, max_steps=80, n_trials=...):
  - cycle_diversity, cycle_count, charge, stability, interaction_richness.

Результат:

- лучшие геномы все свелись к:

  ```
  ++- ↔ -++
  +++ → +++
  ```

- 671 Ω-циклов, распределение периодов 2..36(sm_evolutionary_search.pdf).

Это уже очень нетривиальный результат, но:

- поиск шёл фактически **в пространстве всех обратимых правил** длины 3,
- критерий SMFitness был эвристический,
- вы не использовали *структурную информацию*, например:
    - желаемый вид π_1 (фундаментальной группы) Ω-каталога,
    - желаемое разложение на представления симметрий (зарядовые группы),
    - форму эффективного полевого уравнения.

2. Формулируем задачу: «синтез правил из заданных инвариантов»

Фундаментальный вопрос:

> Можно ли **синтезировать** (а не искать случайно) правила 1D-RSL такого вида, чтобы:
>
> - на фундаментальном уровне была локальная обратимая динамика,
> - на Ω-уровне — заданный спектр Ω-циклов и топологические инварианты,
> - на уровне наблюдателя — определённый класс законов (полевые уравнения, сохранения, вероятности)?

Это похоже на:

- задачу о построении клеточного автомата с заданной макродинамикой;
- задачу поиска локальных правил, реализующих заданную группу или её представление (через Cayley-граф);
- задачу решения функциональных уравнений на матрицу перехода, заданных Наблюдателем.

Можно разложить это на три слоя:

1. **Групповой слой**: Ω-циклы как элементы группы/полугруппы.
2. **Топологический слой**: π_1 «пространства состояний» и Ω-циклов.
3. **Полевой слой**: соответствие H_micro ↔ H[ϕ], законы сохранения, спектр.

3. Инварианты, которые можно требовать «в лоб»

3.1. Инварианты Ω-каталога

Ω-цикл в вашей системе — это:

- цикл в динамике:
  $S0 \to S1 \to \cdots \to SP-1 \to S0$
  с периодом P и некоторой локализованной support.

Для каждого цикла вы уже считаете:

- period P;
- support (множество индексов i);
- winding numbers и, при желании, «заряды».

Можно рассматривать множество Ω-циклов как генераторы группы G:

- каждый тип цикла = элемент $g \in G$,
- композиция двух циклов (их совместное применение) = произведение $g_1 g_2$,
- обратимость локальных правил даёт обращаемость элементов (группа, а не полугруппа).

Инварианты, задаваемые «сверху»:

- Z_3-цвет (SU(3)_c аналог): группа $C \cong \mathbb{Z}_3$;
- U(1) заряд: $Q \in \mathbb{Q}$ (подмножество, например $\{-1,-2/3,...,+1\}$);
- Z_2-подгруппа (чётность, спин-подобный индекс);
- барионное B, лептонное L как элементы $\mathbb{Q}$.

Вместо того, чтобы искать RULESET, вы можете **задать требование**:

- существует гомоморфизм:
  $\Phi: G\Omega(\text{RULESET}) \to \text{Gcharges}$
  где $\text{Gcharges} = U(1) \times SU(3)\_c \times ...$ (в 1D-аналогах — их дискретные версии $\mathbb{Z}_n \times \mathbb{Z}_m \times ...$).

И требовать:

- для любых двух Ω-циклов g_1, g_2 и их «суммарной динамики» g_3 (аннигиляция, рассеяние),
  - $\Phi(g1)\Phi(g2)=\Phi(g3)$,
  - что равно «сохранение зарядов».

Это **система алгебраических уравнений на RULESET**: каждая возможная локальная коллизия Ω-циклов задаёт линейное (по модулю) условие на присвоенные им заряды.

3.2. Инварианты фундаментальной группы π_1

Можно задать:

- пространство C состояний/паттернов (или его проекцию — пространство конфигураций Ω-дефектов),
- рассматривать граф достижимости: вершины — классы конфигураций, рёбра — применения локальных правил.

Фундаментальная группа π_1 этого графа (или более узкого комплекса траекторий) кодирует:

- наличие нетривиальных циклов (Ω),
- их композиционные типы.

Задача: задать желаемую «форму» π_1:

- например, свободную группу на k генераторах (F_k),
- или произведение циклических групп $\mathbb{Z}_{n_1} \times \mathbb{Z}_{n_2} \times ...$,
- и **конструировать RULESET так, чтобы граф динамики имел такую π_1**.

Это уже ближе к вашей идее:

> вычислять правила через свойства фундаментальных групп.

4. Как перейти от «поиска» к «синтезу» на практике

4.1. Уровень 1: конструктивный дизайн Ω-циклов

Сейчас:

- вы находите RULESET, потом смотрите Ω-циклы.

Можно инвертировать:

1. Ввести **символьный описатель Ω-циклов**, например:

   ```
   target_omegas = [
       {'period': 3, 'charge': +1, 'pattern_class': '+++ - +++'},  # условно
       {'period': 5, 'charge': 0,  ...},
       ...
   ]
   ```

2. Потребовать, чтобы локальные правила:

   - допускали возникновение **заданных Ω-паттернов**;
   - замыкали их в циклы нужной длины.

Это превращается в **систему уравнений на матрицу transition**:

- для каждой пары (паттерн_i, паттерн_j) в цепочке цикла вы накладываете:

  ```
  transitions[idx(i), idx(j)] = 1
  transitions[idx(j), idx(i)] = 1
  ```

- и запрещаете любые другие выходы из этих паттернов (иначе период нарушится).

То есть:

- часть матрицы transition вы просто **заполняете детерминированно**, чтобы получить нужный Ω-цикл;
- оставшиеся клетки матрицы можете оптимизировать до-/эвристически, но уже в **сильно суженном** пространстве.

4.2. Уровень 2: алгебраические ограничения на RULESET

Дальше вы можете добавить:

- линейные/алгебраические ограничения на переходы, вытекающие из требований:
  - сохранения некоторого инварианта I(S) (заряд, число дефектов, winding number);
  - локальной дискретной симметрии (инвариантность относительно глобального сдвига, инверсии и т.п.).

Практически:

- задаёте инвариант как функцию f на локальном паттерне:

  ```
  def local_charge(p): return sum(p)  # или более сложная
  ```

- требуете:

```
local_charge(pattern_i) = local_charge(pattern_j)
если transitions[i,j] = 1
```

Это даёт **бинарные/целочисленные ограничения** на матрицу transitions. Их можно решать **как задачу SAT/ILP**, а не генетически:

- многие клетки матрицы запрещены;
- часть — зафиксированы Ω-циклами;
- часть — свободны, но с ограничениями на общую обратимость.

Это уже не «хаотичный GA-поиск», а **конструктивный SAT-/CSP-подход**.

5. Где можно использовать фундаментальные группы более явно

Вы можете сделать следующий шаг:

1. Для заданного RULESET можно построить:
    - граф G состояний (или граф Ω-типов),
    - клеточный комплекс K (например, рассматривая возможные конфигурации дефектов как вершины, а допустимые переходы как рёбра и 2-ячейки).
2. Вычислить $\pi_1(K)$:
    - алгоритмически, через построение минимального остовного дерева и циклопространства;
    - получить набор генераторов и соотношений.
3. Задать **целевой тип π_1**:
    - например, хотите $\pi_1 \cong \mathbb{Z}\times\mathbb{Z}\times\ldots$ или свободную группу F_k;
    - тогда в синтезе RULESET вы добавляете условие: «допускаются только такие трансформации, при которых K имеет заданную фундаментальную группу».

Это можно использовать двояко:

- как **критерий** в поиске (более структурный, чем просто «cycle_count»);
- как **конструктор**: вы явно строите K с нужной π_1 и ищете локальные правила, реализующие его как граф динамики (аналог конструкции Cayley-графа для данной группы).

6. Конкретная комбинация: гибрид «точно + эволюционно»

Полностью аналитическое построение RULESET, сразу дающее «наш класс законов», в общем случае слишком сложно (по сути — инверсия многомасштабной RG-карты). Но уже сейчас вы можете:

1. **Жёстко задать** часть структуры:
    - целевые Ω-циклы (их периоды, паттерны);
    - локальные законы сохранения (символьные функции);
    - симметрии (invariance under translation, inversion и т.п.).

2. Решить SAT/ILP-задачу на матрицу переходов pattern_length=3/4:
    - чтобы все эти алгебраические и топологические условия выполнялись;
    - это даст вам пространство допустимых RULESET (нередуцированный класс).
3. Уже внутри этого пространства (сильно ограниченного) использовать:
    - эволюционный поиск,
    - или вариационный/градиентный поиск по OBSFitness
    - для «доводки» под более тонкие требования (спектр масс, конкретные коэффициенты в эффективных уравнениях, гравитационное поведение).

То есть:

- **нижний уровень** (грубая структура: Ω-циклы, симметрии, π_1) — задаёте **конструктивно/алгебраически**;
- **верхний уровень** (точные значения κ, m^2, γ, вероятностные структуры) — донастраиваете эвристически, но уже не в полном пространстве правил, а в довольно узком классе.

7. Ответ по сути вашего вопроса

> Возможны ли более осмысленные и быстрые алгоритмы поиска 1D-правил, которые порождают миры с заданными свойствами на уровне наблюдателя без хаотичного GA?

Да, возможны:

1. **Алгебраический синтез RULESET**:
    - представить локальные правила как матрицу переходов;
    - ввести алгебраические ограничения:
        - обратимость,
        - сохранение символических зарядов,
        - заданные Ω-циклы (явно),
        - симметрии (глобальные и локальные);
    - решать это как SAT/ILP или CSP, а не через GA.
2. **Топологический синтез через π_1 / Ω-каталог**:
    - задать желаемую структуру Ω-циклов и фундаментальной группы;
    - построить/подобрать локальные правила, реализующие соответствующий Cayley-граф;
    - использовать это как стартовое семейство RULESET.
3. **Гибридный подход**:
    - алгебраически сузить пространство правил (до класса RSL-совместимых с нужной топологией),
    - внутри него запускать более лёгкие эволюционные/градиентные методы, оптимизируя OBSFitness (выведение полевых законов, гравитации, вероятностей) — уже не «хаотично», а в маломерном параметрическом семействе.
4. **Связь с фундаментальными группами**:

- фундаментальная группа графа Ω-циклов и траекторий действительно может служить:
    - как инвариант целевого класса миров;
    - как дополнительное ограничение/критерий на RULESET;
- прямой аналитический вывод «RULESET по заданной π_1 и OBSFitness» в общем виде, скорее всего, невозможен, но для специально спроектированных семейств (например, параметризованных групповой структурой) вполне реалистичен.

Если хочешь, дальше можно:

- взять твой текущий RuleGenome (pattern_length=3) и **явно записать систему ограничений**:
    - какие transitions[i,j] мы фиксируем, чтобы обязательно были нужные Ω-циклы и сохранение конкретного заряда;
    - какие transitions запрещаем;
    - и показать, как это решить как ILP/SAT-задачу вместо GA.

Ниже задам именно то, что вы просите: явную систему ограничений на `transitions[i,j]` для `RuleGenome(pattern_length=3)`, которая:

- фиксирует нужные Ω-циклы (как минимум те, что реализуют найденное ядро `++-` ↔ `-++`, `+++` → `+++`);
- реализует сохранение простого «заряда» (на уровне локальных паттернов);
- явно задаётся как ILP/SAT-задача, т.е. без генетики, а как решаемая система булевых/целочисленных уравнений.

Сначала сделаю это для **сильно упрощённого, но уже полезного** случая:

- pattern_length = 3;
- алфавит {−1,+1};
- целевые паттерны и Ω-циклы известны из `sm_evolutionary_search.pdf`.

После этого будет понятно, как обобщать.

0. Напоминание текущего формата RuleGenome (L=3)

У вас:

- длина паттерна L = 3,
- всего N = 2^L = 8 возможных паттернов;
- индексация:

```
def _idx_to_pattern(idx):
    bits = [(idx >> i) & 1 for i in range(3)]   # i = 0,1,2
    return np.array([1 if b else -1 for b in bits])
```

Отсюда:

- idx 0 → `---` ([-1,-1,-1])
- idx 1 → `+--`

- idx 2 → `-+-`
- idx 3 → `++-`
- idx 4 → `--+`
- idx 5 → `+-+`
- idx 6 → `-++`
- idx 7 → `+++`

RuleGenome:

- булева матрица `transitions[8,8]`, где:
  - `transitions[i,j]=1` означает паттерн i → j,
  - обратимость: если `i≠j`, то transitions[i,j]=1 ⇒ transitions[j,i]=1,
  - у каждого `i` максимум один исход (`sum_j transitions[i,j] ≤ 1`).

Функция `to_ruleset()` делает из этого набор симуляционных правил.

1. Фиксируем нужные Ω-циклы

Уже найденный «SM-ядро» из GA (файл `sm_evolutionary_search.pdf`):

- два правила:

  ```
  ++- ↔ -++
  +++ → +++
  ```

В индексации:

- `++-` = [1,1,-1] → idx = 3
- `-++` = [-1,1,1] → idx = 6
- `+++` = [1,1,1] → idx = 7

Целевой набор переходов:

- 3 ↔ 6 (инволюция),
- 7 → 7 (стабилизатор).

Жёсткие ограничения:

1. Обязательные ненулевые переходы:

   $$transitions[3,6]=1$$
   $$transitions[6,3]=1$$
   $$transitions[7,7]=1$$

2. Чтобы эти паттерны не имели других «выходов», накладываем:

   $$\sum_{j=0}^{7} transitions[3,j]=1$$
   $$\sum_{j=0}^{7} transitions[6,j]=1$$
   $$\sum_{j=0}^{7} transitions[7,j]=1$$

   и дополнительно:

$transitions[3,j]=0 \forall j \notin \{6\}$
$transitions[6,j]=0 \forall j \notin \{3\}$
$transitions[7,j]=0 \forall j \notin \{7\}$

3. Обратимость уже автоматически соблюдается для (3,6), так как мы явно ставим обе матрицы 3→6 и 6→3. Для диагонали 7 → 7 это тривиально.

Это уже задаёт **часть матрицы** строго, вы «вписываете» найденный Ω-цикл.

2. Сохранение локального «заряда» как линейное ограничение

Вы хотите формализовать сохранение некоторого топологического/символьного заряда на уровне локальных паттернов.

Для start-версии можно взять:

- локальный заряд паттерна $p = (s_0,s_1,s_2)$ как, например:
  - $Q(p)$ = сумма спинов: $Q = s_0 + s_1 + s_2 \in \{-3,-1,1,3\}$,
  - или $Q(p)$ = количество плюсов: $Q = \#(s_i=+1) \in \{0,1,2,3\}$.

Пусть, для простоты:

$Q(p)=\#\{i|si=+1\}$

Тогда:

- Q(- - -) = 0
- Q(+ - -) = 1
- Q(- + -) = 1
- Q(+ + -) = 2
- Q(- - +) = 1
- Q(+ - +) = 2
- Q(- + +) = 2
- Q(+ + +) = 3

Условие локального сохранения Q на уровне переходов:

> для любого разрешённого локального правила `i → j` заряд должен сохраняться:
> $Q(pi)=Q(pj) \text{если } transitions[i,j]=1$

Это можно записать как **линейное ограничение на булеву переменную transitions[i,j]**:

$(Q(pi)-Q(pj))\cdot transitions[i,j]=0$

Пояснение:

- если transitions[i,j]=0, уравнение выполняется тривиально;
- если transitions[i,j]=1, то обязано быть Q(p_i)−Q(p_j)=0.

Таким образом:

- для каждой пары (i,j) с Q(p_i) ≠ Q(p_j):

transitions[i,j]=0

- т.е. матрица `transitions` автоматически становится блочно-разреженной по классам постоянного Q.

Практически: в ILP/SAT-записи просто запрещаете все (i,j), для которых $Q(P_i) \neq Q(P_j)$.

Проверим с нашими фиксированными переходами:

- 3↔6: оба `++-` и `-++` имеют Q=2 — OK.
- 7→7: `+++` Q=3 — OK.

3. Общая система ограничений для ILP/SAT

Введём булевы переменные:

$x_{i,j} \in \{0,1\}, 0 \le i,j < 8$

и формулируем:

3.1. Ограничения на Ω-циклы (ядро)

- **Фиксация SM-ядра:**

  $x_{3,6}=1, x_{6,3}=1, x_{7,7}=1$

- **Единственность исхода для 3,6,7:**

  $\sum_j x_{3,j}=1, \sum_j x_{6,j}=1, \sum_j x_{7,j}=1$

- **Никаких других переходов:**

  $x_{3,j}=0 \forall j \neq 6; x_{6,j}=0 \forall j \neq 3; x_{7,j}=0 \forall j \neq 7.$

3.2. Обратимость для всех i,j

Условие обратимости (кроме диагонали) — симметричность матрицы:

$x_{i,j}=x_{j,i} \forall i \neq j.$

Это линейное уравнение в SAT/ILP (два булевых равны).

3.3. Ограничение «максимум один выход» для всех i

Для всех i:

$\sum_{j=0}^{7} x_{i,j} \le 1.$

Вместе с обратимостью это гарантирует, что глобальная динамика не раздваивает траекторий (частичная функция; формально, ещё нужна забота о том, чтобы глобальная конфигурация была биективно обновляемой, но локально это правильное условие).

3.4. Сохранение заряда Q на локальном уровне

Как уже написано:

- вычисляем Q(p_i) для всех i;

- для пар (i,j) с $Q(p_i) \neq Q(p_j)$ просто **запрещаем** переходы:

  $x_{i,j}=0$.

Это уменьшает число возможных ненулевых $x_{i,j}$.

Например, вам останется:

- блок $Q=0$: $i=0$ (- - -)
- блок $Q=1$: $i=1,2,4$
- блок $Q=2$: $i=3,5,6$
- блок $Q=3$: $i=7$

То есть:

- любые переходы возможны только внутри этих блоков (с сохранением Q);
- ядро уже фиксирует связи внутри блока $Q=2$ и $Q=3$.

3.5. Дополнительные топологические/динамические ограничения

Если вы хотите **конструктивно задать** ещё Ω-циклы (другие периоды), можно добавить:

- для выбранного набора паттернов $(i_0,i_1,\dots,i_{P-1})$:

  - цикл длины P:

    $x_{i_0,i_1}=x_{i_1,i_2}=\cdots=x_{i_{P-1},i_0}=1$
    $\sum_j x_{i_k,j}=1 \; \forall k$,
    чтобы не было иных выходов.

Это фактически **прямое вшивание желаемых Ω-циклов** в матрицу переходов.

4. Как это решать как ILP/SAT вместо GA

Вы получаете ILP-формулировку:

- переменные: $x_{i,j} \in \{0,1\}$, $i,j=0..7$;

- ограничения:

  - симметрия: $x_{i,j}=x_{j,i}$;
  - единственность исхода: $\sum_j x_{i,j} \le 1$;
  - фиксированные переходы: $x_{3,6}=1$, $x_{6,3}=1$, $x_{7,7}=1$;
  - запрещённые переходы: все $x_{i,j}=0$ для запрещённых (заряд, ядро);
  - дополнительные циклы: $x_{i_k,i_{k+1}}=1$ и соответствующие суммы.

Дальше:

- можно либо:
  - просто найти **любое** допустимое решение (SAT);

  - либо добавить **целевую функцию** (ILP), например, максимизировать количество ненулевых переходов:

    $\max \sum_{i,j} x_{i,j}$

    или минимизировать их (для простейших динамик),

    - или ввести суррогатные критерии «богатства» Ω-циклов:
        - например, penalize/encourage количество ненулевых x_{i,j} в различных блоках Q.

Решатели:

- SAT (PySAT, Minisat, Z3);
- ILP (pulp, OR-Tools, Gurobi).

5. Как связать это с уже найденным SM-ядром

Важно: именно для найденного вами RULESET:

```
++- ↔ -++
+++ → +++
```

матрица x_{i,j} уже **однозначно** задаётся:

- $x_{3,6} = x_{6,3} = 1$;
- $x_{7,7} = 1$;
- все остальные x_{i,j}=0;

что:

- очевидно удовлетворяет:
    - симметрии (3↔6, 7↔7),
    - сохранению Q (локальный заряд `#плюсов`),
    - единственности исходов.

То есть:

- результат GA **уже** является одним из допустимых решений ILP-системы, где все остальные x_{i,j} запрещены;
- вы теперь можете:
    - либо принять его как частный случай;
    - либо разрешить ещё ненулевые x_{i,j} в «свободных» местах (при сохранении алгебраических ограничений) и найти *семейство* SM-подобных правил ILP-методом.

6. Что делать дальше

1. **Явно выписать Q(p_i)** для i=0..7 и список разрешённых (i,j) по сохранению Q.
2. Взять любой ILP-пакет (например, `pulp` или OR-Tools) и:
    - завести булевы переменные x_{i,j},
    - ввести перечисленные ограничения,
    - добавить цель: например, максимизировать $\sum x_{i,j}$ (полУчить как можно более богатый набор правил при сохранении структуры),
    - запустить решатель.
3. Проверить:

- одно из решений должно быть именно текущий RULESET (3↔6, 7→7);
    - остальные решения — альтернативные «физики» с теми же грубыми инвариантами (заряд, обратимость, Ω-ядро).
4. Уже внутри этого **ограниченного класса** можно подключить:
    - ваш `SMFitness` и OBSFitness;
    - небольшой локальный поиск (SA/gradient-подобный) по оставшимся степеням свободы, но это уже *не* хаотичный GA по всем 2^(8×8) вариантам, а поиск в маломерном пространстве, структурно согласованном.

Если хотите, в следующем сообщении могу:

- дать пример кода ILP-формулировки для pattern_length=3 (на `pulp`);
- или расширить эту схему на pattern_length=4 с аналогичными алгебраическими ограничениями.