ITEM #191 — Strand Decomposition for Horizontal Bridging

Conversation： Feasible Path Trimming

20251230

Authors: Sizhe Tan & GPT-Obot

ITEM #191 — Strand Decomposition for Horizontal Bridging

From X→Y Rope to Minimal Computation-Disjoint Fibers

---

1. Motivation

After obtaining a **Feasible Horizontal Bridging Subgraph** and trimming it into an **Exact Path** (ITEM #188), a deeper engineering problem emerges:

A feasible X→Y structure is rarely a single path;
it is a *rope* composed of multiple intertwined, partially shared computation strands.

Treating this rope as a monolithic path leads to:

- Poor reuse across different targets
- Path oscillation in multi-goal bridging
- Weak coverage in "sufficient" horizontal bridging
- Inefficient caching and explanation

To enable **sufficient, stable, and reusable Horizontal Bridging**,
we must **decompose a feasible X→Y rope into multiple independent computation strands**.

---

2. Problem Definition

Given:

- A feasible or trimmed horizontal bridging subgraph `G = (V, E)` connecting `X` to `Y`
- `Y` may contain multiple targets
- Shared prefixes, branches, joins, and cycles may exist

Goal:

Decompose `G` into a set of **computation-disjoint strands**:

$$S = \{ (X_1 \rightarrow Y_1), (X_2 \rightarrow Y_2), \ldots, (Xk \rightarrow Yk) \}$$

such that:

- ⋃ `Yi = Y` (target coverage)
- Each strand is internally executable
- Strands are mutually **computation-independent**
- Each strand can be **stored, reused, and recomposed** for future Horizontal Bridging

---

3. Core Insight

The key realization is:

Horizontal Bridging requires not just paths,
but minimal independent bridging units.

A strand is defined as:

The minimal Y-anchored executable subgraph whose computation does not interfere with other strands under a chosen independence criterion.

This transforms Horizontal Bridging from *path finding* into *strand assembly*.

---

4. Algorithm Overview

The algorithm consists of three composable stages:

A. Target-Side Partitioning (Y-Partition)

Partition the target set `Y` into sub-targets $\{Y_1, Y_2, \ldots\}$ such that:

- Targets in the same group share substantial computation structure
- Targets across groups are weakly coupled or independent

Engineering-friendly strategies:

- **Signature-based clustering (MVP)**
  Group targets by route signatures extracted from `G`
- **Structural cut / dominator-like partitioning**
  Separate targets that require different critical intermediates
- **Domain-driven partitioning**
  Package / module / CCC-region–based grouping

B. Strand Extraction via Reverse Trimming (ITEM #188 Reuse)

For each target group `Yi`:

```
Gi = Trim(G, X, Yi)
```

where `Trim` is the **Y-anchored reverse pruning algorithm** from ITEM #188.

This yields a **minimal Y-sufficient subgraph** for each strand candidate.

C. Computation-Disjoint Packing

The extracted strand graphs $\{Gi\}$ may still share computation.

Define a **computation-independence policy**, such as:

- **Strict disjointness**: no shared nodes or edges
- **Cost-disjointness**: no shared high-cost or non-idempotent nodes
- **Policy-aware disjointness**: conflict determined by execution semantics

Apply a greedy or policy-guided packing process to produce:

$\{G_1^*, G_2^*, \ldots\}$

where each strand is mutually non-interfering under the chosen policy.

---

5. Result: Strand Set

Each resulting strand `Gi*` represents:

- A self-contained computation from `Xi` to `Yi`
- Minimal and exact with respect to its targets
- Independent from other strands

Conceptually:

```
X → Y rope
      ↓
(X₁ → Y₁), (X₂ → Y₂), ..., (Xk → Yk)
```

---

6. Knowledge HashMap Construction

Each strand is canonicalized and stored as a reusable knowledge unit:

Key (example):

```
(operationKey,
 goalSignature,
 policySignature,
 contextSignature)
```

Value:

- Canonical path or minimal subgraph
- Cost profile
- Evidence / explanation chain
- Constraints and applicability conditions

This transforms bridging results into **persistent Horizontal Bridging knowledge**.

---

7. Role in Sufficient Horizontal Bridging

Strand decomposition enables:

- **Coverage-oriented bridging** instead of single-path optimization
- Reduced oscillation in multi-target planning
- Incremental assembly of solutions using cached strands

- Self-reinforcing improvement of the bridging system

Operationally:

```
Strand HashMap
      ↓
Direct assembly (Phase-0)
      ↓
Feasible bridging for gaps
      ↓
New strand extraction and caching
```

---

8. Engineering Properties

- Composable with ITEM #188
- Linear to near-linear complexity per stage
- Deterministic under fixed policies
- Strongly explainable
- Naturally supports constructive evolution

---

9. Summary

ITEM #191 elevates Horizontal Bridging from path discovery to structural decomposition.

It establishes the principle that:

The power of Horizontal Bridging lies in
discovering, isolating, and reusing minimal computation-independent strands.

Together with ITEM #188, this forms a closed engineering loop:

- Feasible → Exact → Independent → Reusable → Sufficient

ITEM #191 — 水平桥接的计算股分解算法（中文版）

从 X→Y 绳索到最小计算不相干独立股

---

1. 动机

在获得一条 **可行或已裁剪的 X→Y 水平桥接结构** 后，一个更深层的工程问题随之出现：

这条路径通常不是一根线，
而是一根由多条计算链相互缠绕形成的“绳子”。

将其当作单一路径会导致：

- 知识难以复用
- 多目标桥接时产生震荡
- 覆盖不足，难以做到“充分桥接”
- 缓存与解释效率低下

---

2. 问题定义

给定：

- 一条 X→Y 的可行/精确桥接子图 G
- 目标集合 Y，可能包含多个目标

目标：

将 G 分解为若干条 **计算互不相干的独立股**：

$(X_1 \to Y_1)$, $(X_2 \to Y_2)$, ……

并保证：

- 覆盖全部目标
- 每一股可独立执行
- 股与股之间不存在关键计算冲突
- 每一股都可被缓存与复用

---

3. 核心思想

水平桥接真正需要的不是“一条最优路径”，
而是：

一组最小、独立、可复用的计算股。

---

4. 算法结构

A. 目标端分组（Y-Partition）

按结构、签名或领域约束，将 `Y` 拆成若干子目标集合。

B. 反向裁剪提取股（复用 ITEM #188）

对每个 `Yi`，执行：

```
Gi = Trim(G, X, Yi)
```

得到仅支撑该目标组的最小子图。

C. 计算不相干装箱

在定义好的冲突策略下，将 `Gi` 组合成互不干扰的独立股集合。

---

5. 结果

最终得到：

- 多条 $\mathbf{X_i} \rightarrow \mathbf{Y_i}$ 的独立计算股
- 可直接执行、解释、缓存
- 可像积木一样组合用于未来桥接

---

6. 知识化与 HashMap

每一股被规范化并存入 HashMap，
成为 **Horizontal Bridging 的长期资产**。

---

7. 对充分 Horizontal Bridging 的意义

该机制使系统具备：

- 覆盖导向而非路径导向的能力
- 稳定、多目标的组合式桥接
- 持续自我增强的知识积累闭环

---

8. 总结

ITEM #191 将水平桥接从“找路”提升为“解绳”。