Contents

1 Chapter 6 Winning Parameters: LinUCB vs Thompson Sampling

This document records the **Algorithm Selection Principle** discovered in Chapter 6:

> **Algorithm selection depends on feature quality.** - Clean/Oracle features → LinUCB (precise exploitation) - Noisy/Estimated features → Thompson Sampling (robust exploration)

1.1 Configuration A: Rich Features + Oracle Latents (LinUCB Wins)

When you have access to true user latent preferences (oracle features), LinUCB's precise exploitation wins:

```
python scripts/ch06/template_bandits_demo.py \
    --features rich \
    --rich-regularization blend \
    --n-static 2000 \
    --n-bandit 20000 \
    --world-seed 20250322 \
    --bandit-base-seed 20250349 \
    --prior-weight 50 \
    --lin-alpha 0.2 \
    --ts-sigma 0.5
```

1.1.1 Expected Results (Oracle Latents)

Policy	GMV	ΔGMV vs Static
Static-Premium	7.11	baseline
LinUCB	**9.42**	**+32.5%**
Thompson Sampling	9.39	+32.1%

Winner: LinUCB (marginally, +0.4 percentage points)

Why LinUCB wins: With clean oracle features, the linear model assumption holds nearly perfectly. LinUCB's UCB exploration bonus shrinks precisely as uncertainty decreases, enabling pre-

cise exploitation. Note that TS also performs excellently with oracle features—the real differentiation emerges under feature noise (see Estimated Latents below).

1.2 Configuration B: Rich Features + Estimated Latents (TS Wins)

In production, you don't have oracle latents—you must estimate user preferences from behavior. With noisy estimated features, Thompson Sampling's robust exploration wins:

```
python scripts/ch06/template_bandits_demo.py \
    --features rich_est \
    --n-static 2000 \
    --n-bandit 20000 \
    --world-seed 20250322 \
    --bandit-base-seed 20250349 \
    --prior-weight 50 \
    --lin-alpha 0.2 \
    --ts-sigma 0.5
```

1.2.1 Expected Results (Estimated Latents)

Policy	GMV	ΔGMV vs Static
Static-Premium	7.11	baseline
LinUCB	7.52	+5.8%
Thompson Sampling	**9.31**	**+31.0%**

Winner: Thompson Sampling by 25 percentage points

Why TS wins: With noisy estimated features, LinUCB can lock into suboptimal templates based on spurious correlations. Thompson Sampling's posterior sampling maintains perpetual exploration variance, hedging against feature noise and avoiding premature convergence.

1.3 The Algorithm Selection Principle

Feature Quality	Winner	Margin	Recommendation
Oracle (clean)	LinUCB	+0.4 pts	Use when features are direct measurements or carefully validated
Estimated (noisy)	TS	+25 pts	**Default for production**

Key insight: With oracle features, both algorithms perform nearly identically (LinUCB 9.42, TS 9.39). The dramatic differentiation emerges **only under feature noise**—TS maintains +31% lift while LinUCB drops to +6%.

Production implication: Real e-commerce systems have noisy estimated features—inferred from clicks, aggregated from proxies, delayed from behavior logs. **Default to Thompson Sampling in production.**

1.4 Hyperparameter Reference

Parameter	Value	Notes
`--prior-weight`	50	Pseudo-count for static priors
`--lin-alpha`	0.2	LinUCB exploration bonus (conservative)
`--ts-sigma`	0.5	Thompson Sampling posterior std
`--rich-regularization`	blend	Regularization mode for rich features
`--n-static`	2000	Episodes for static baseline evaluation
`--n-bandit`	20000	Episodes for bandit training

1.5 Running the Full Compute Arc

To reproduce the complete three-stage narrative:

```
python scripts/ch06/ch06_compute_arc.py \
    --n-static 2000 \
    --n-bandit 20000 \
    --base-seed 20250322 \
    --out-dir docs/book/drafts/ch06/data \
    --prior-weight 50 \
    --lin-alpha 0.2 \
    --ts-sigma 0.5
```

This runs all three stages: 1. **Simple features** (failure mode) 2. **Rich features + oracle latents** (LinUCB wins) 3. **Rich features + estimated latents** (TS wins)

And produces JSON summaries + comparison tables demonstrating the Algorithm Selection Principle.

1.6 References

- Chapter 6 main narrative: `docs/book/drafts/ch06/discrete_template_bandits.md`
- Lab exercises: `docs/book/drafts/ch06/exercises_labs.md`
- Compute arc script: `scripts/ch06/ch06_compute_arc.py`