A GRASP metaheuristic for a territorial design problem in financial institutions

Eduardo Salazar Treviño, Roger Z. Ríos Mercado, Diana L. Huerta Muñoz

Department of Mechanical and Electrical Engineering
Universidad Autónoma de Nuevo León
San Nicolás de los Garza, N.L.

XI Congreso de la Sociedad Mexicana de Investigación de Operaciones
October 19th, 2023

Outline

Territorial Design Problem

Problem formulation

- Classic problem, also termed as "Districting Problem" [4].
- Organizes a set B of *basic units* (BUs) and a set S of territory centers into p larger territories or districts.
- Key constraints:
  - Spatial: compactness, contiguity.
  - Planning: balance of activities (e.g., sales, workload).
- Aim: Achieve compact territories using dispersion measures from problems such as p-center or p-median while balancing diverse metrics.

Microfinancial Institutions and Applications

Problem Formulation

- Motivated by previous TDP application to a real-life microfinancial institution [1].
- Role of Microfinance Institutions (MFIs):
  - Alleviate income inequality and poverty.
  - Provide credit and financial services to high-risk markets.
  - Typically avoided by commercial banks due to high risk and low profit.

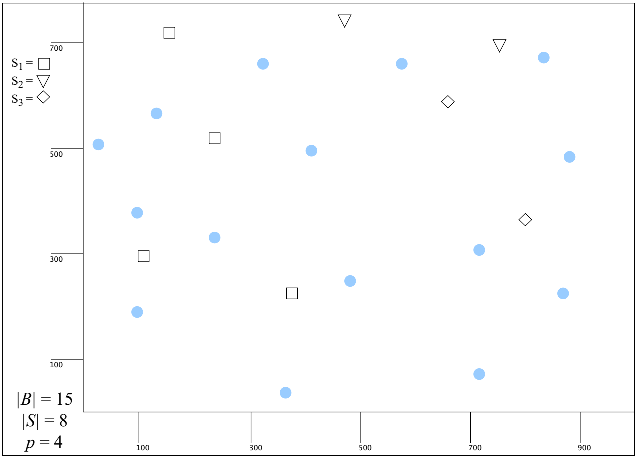
Microfinancial Institutions' Special Requirements

Problem Formulation

- TDP's role: Identify potential branch office locations and allocate clients to their office.
- Importance of balance:
    - Distributed centers across different host's business lines.
    - Activity metrics evenly spread (e.g., loan amounts, client count).
    - Manage loan risks due to volatility.
- Goal: Minimize distance between BUs and institution branches.

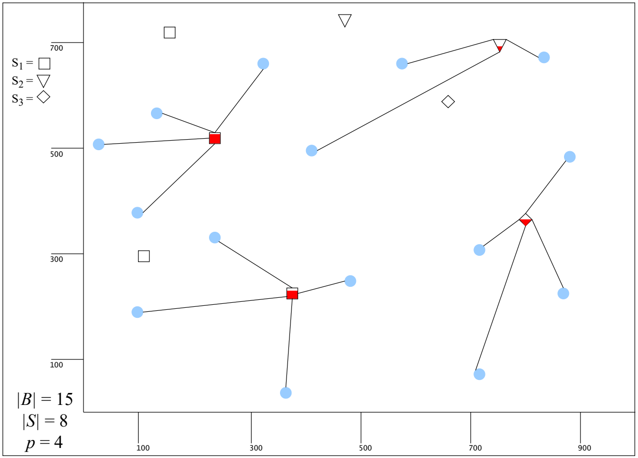
Financial institution territory example

Illustrative example



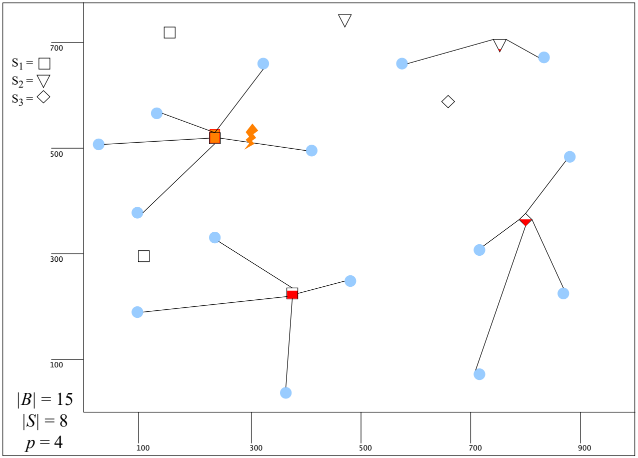
Financial institution territory example

Feasible solution example



Financial institution territory example

Unfeasible solution example



Motivation and Purpose

- **Model Differences:**
  - Incorporates risk balancing as a constraint.
  - Previous model's objective: Retain original territorial design.
  - Current model omits contiguity as a constraint.
- **Model Features:**
  - Comparable to vertex p-center problem with capacity constraints.
  - Given uncapacitated vertex p-center is NP-hard [3], this TDP is also NP-hard.
  - Emphasizes potential of heuristic approaches for resolution.

Combinatorial Model

Problem Formulation

Sets and Parameters:

- S: set of territory centers, B: set of BUs
- d_{ij} : distance between center i and BU j
- $M = \{1, 2, ..., m\}$: number of activities to measure
- $K = \{1, 2, ..., k\}$: number of center types.
- T_{ik}: 1 if center i is of type k, 0 otherwise
- μ^i_m: target of activity m at center i, $\forall m \in M$
- v^j_m: activity measure m at BU j, $\forall m \in M$
- t_m: tolerance of activity m, $\forall m \in M$
- r_j : risk value at BU j
- β_i : risk threshold at center i
- L_k, U_k: bounds for centers in T_ik, $\forall k \in K$
- p: number of centers to be used

Combinatorial model

Problem formulation

Decision Variables

- $Y_i = 1$ if center i is used; $= 0$ otherwise.
- $X_{ij} = 1$ 1 if BU j is assigned to center i; $= 0$ otherwise.

Mathematical programming Model

Formulation

Objective Function:

$$\mathbf{min} \sum_{i \in S, j \in B} X_{ij} D_{ij} \tag{1}$$

Subject to:

$$\sum_{i \in S} X_{ij} = 1, \quad \forall j \in B \tag{2}$$

Single assignment of a BU j to a center i

$$X_{ij} \leq Y_i, \quad \forall i \in S, \quad j \in B \tag{3}$$

Can only assign BUs to centers that are open

$$Y_i \mu_m^i (1 - t^m) \leq \sum_{j \in B} X_{ij} v_j^m \leq Y_i \mu_m^i (1 + t^m), \quad \forall i \in S \tag{4}$$

Activity measures for each territory must be within a tolerance range

Mathematical programming Model

Formulation

$$L_k \leq \sum_{i \in S} Y_i T_{ik} \leq U_k, \quad k \in \{1 \dots 5\} \tag{5}$$

The selected centers' types must respect the lower and upper bound for each type

$$\sum_{i \in S} Y_i = P \tag{6}$$

The number of centers to be opened must be equal to p

$$\sum_{j \in B} X_{ij} r_j \leq \beta_i, \quad \forall i \in S \tag{7}$$

The risk measure of each territory must not exceed the risk threshold

Proposed heuristic: Greedy Randomized Adaptive Search Procedure

```
Input: $p, \alpha_l, \alpha_a, i_{max}$, Instance
Output: $A^* =$ Solution
 1: $A^* \leftarrow \emptyset$
 2: $f^* \leftarrow \infty$
 3: while $max\_iters > 0$ do
 4:     $RCL \leftarrow$ BuildRCL($\alpha_l, \alpha_a, p$, Instance)
 5:     $X, Y \leftarrow$ Construct($RCL$, $p$)
 6:     $X, Y \leftarrow$ LocalSearch($X, Y,$, $p$, Instance)
 7:     $A \leftarrow (X, Y)$
 8:     if $f(A) < f^*$ then
 9:         $f^* \leftarrow f(A)$
10:         $A^* \leftarrow A$
11:     end if
12:     $max\_iters \leftarrow max\_iters - 1$
13: end while
```

Construction Phase

Constructive Heuristic Phases:

1. **Location Phase:**
   - Determines which p centers to use from available locations.
   - Returns the decision variable vector Y.
2. **Allocation Phase:**
   - Allocates all BUs to a corresponding center.
   - Returns the decision variable matrix X.

Construction Phase

Location Heuristics

- *p*-**dispersion Problem:**
  - Objective: Choose the *p*-most disperse centers from S.
  - Constraints: Must respect L_k and U_k.
  - Approach: Use best current polynomial-time heuristic [2].
- **Semi-Linear Programming:**
  - Relax integrality requirement of X; solve relaxation
  - Outcome: Provides Y with integer values.

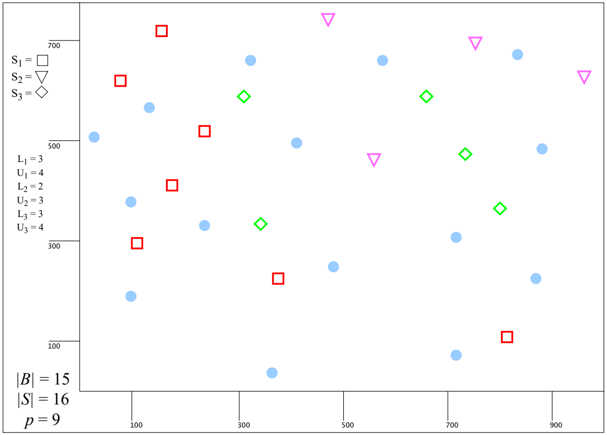
Construction phase

p-dispersion problem

```
Input: $d$, $p$, $S_k$, $L_k$, $U_k$, $S$
Output: $Y$
 1: Compute distance matrices, $\forall k \in S_k$
 2: for $k \in S_k$ do
 3:    PDISP_SIMPLE(distance matrix of $k$, $L_k$, $|S_k|$)
 4:    Add to $Y$ the solution provided for each $k$
 5: end for
 6: while $|Y| < p$ do
 7:    for node in $S$ do
 8:       if node $\notin Y$ then
 9:          Determine type of node in $S_k$
10:          Check if $U_k$ for this node type is reached; if so, skip this node
11:          Compute the minimum distance from node to all nodes in $Y$
12:          Store this minimum distance.
13:       end if
14:    end for
15:    Select the node with the maximum minimum distance
16:    Add this node to $Y$
17: end while
18: return $Y$
```

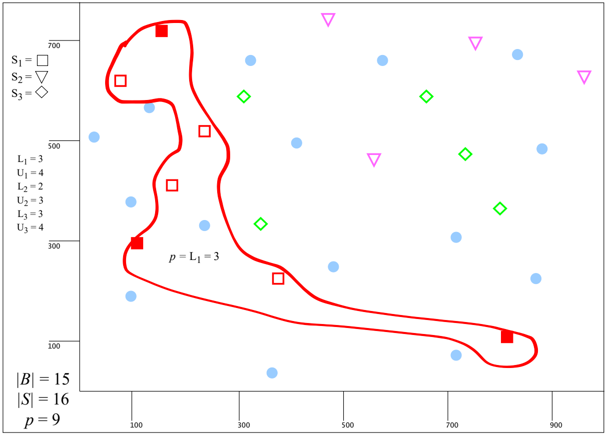
Construction phase

p-dispersion problem heuristic example



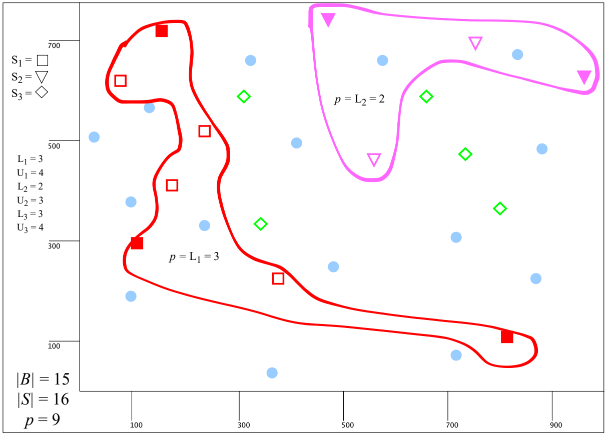
Construction phase

p-dispersion problem heuristic example



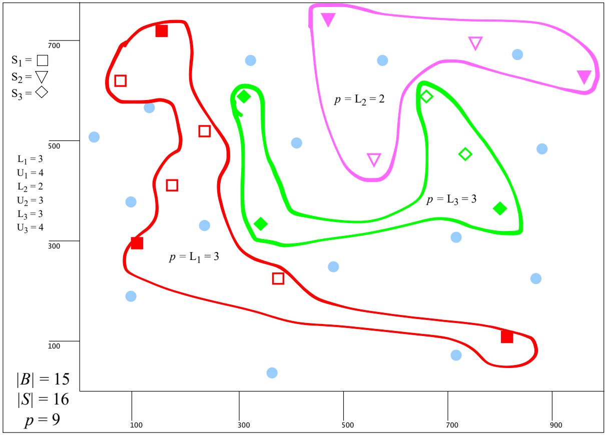
Construction phase

p-dispersion problem heuristic example



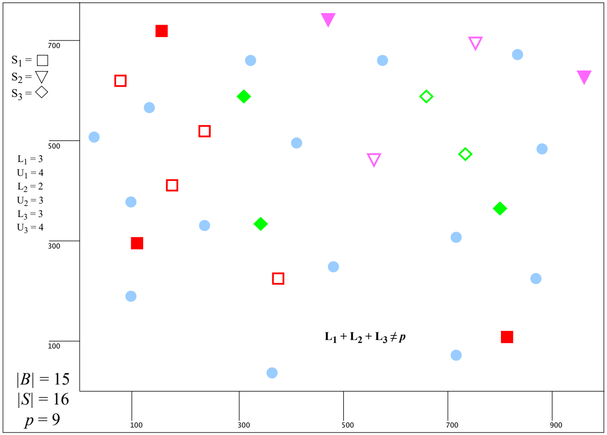
Construction phase

p-dispersion problem heuristic example



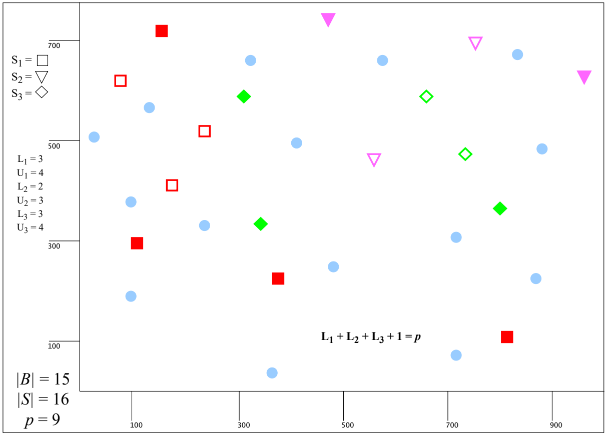
Construction phase

p-dispersion problem heuristic example



Construction phase

p-dispersion problem heuristic example



Construction Phase

Allocation Heuristics

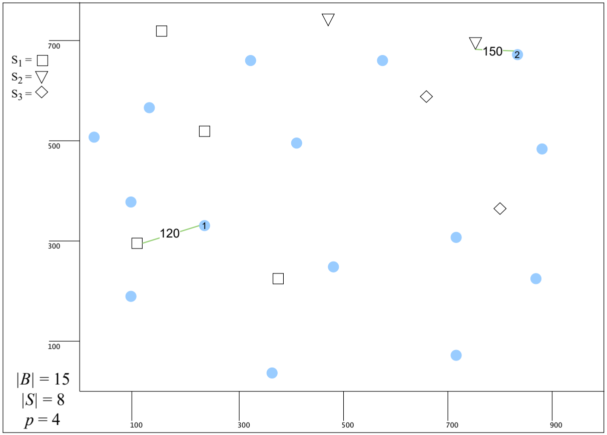
Given the constraints, simply choosing the nearest assignment for each BU isn't always feasible. Rather than minimal distances, we compute the opportunity cost for all BUs, giving priority to those with the highest opportunity cost.

- **Opportunity Cost Approach:**
  - Compute and prioritize based on opportunity cost.
- **Enhanced Approach:**
  - Leverage data structures to refine opportunity cost computations.

Opportunity Cost

Example

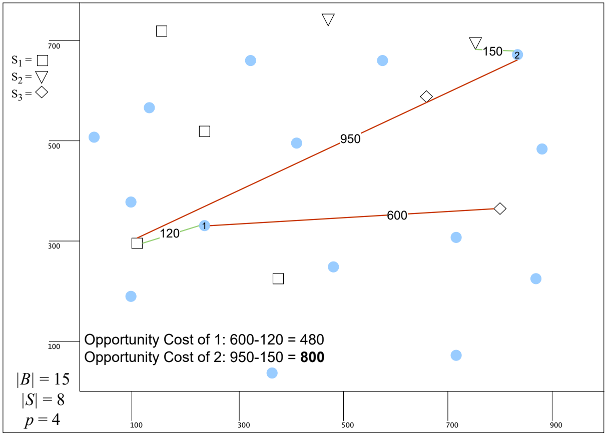
Let us pick two arbitrary BUs, obtaining their nearest center:



Opportunity Cost

Explanation

Get the farthest possible allocation and compute the opportunity cost, selecting the largest:



Opportunity Cost of 1: 600-120 = 480
Opportunity Cost of 2: 950-150 = **800**

Opportunity Cost: Approaches and Data Structures

Explanation and Efficiency Improvements

Explanation:

- Prioritize BUs by their opportunity cost, allocating those most affected by constraints first.
- Two approaches:
  1. Calculate costs for all centers per BU, generating a matrix as large as d.
  2. Compute only the largest opportunity cost and store it in a priority queue.

Data Structures:

- Identified performance bottlenecks in memory allocations, traversals, and updates.
- Enhanced with a precomputed opportunity cost priority queue and a nearest assignment dictionary for BUs.
- Improved worst-case scenario from $\mathcal{O}(|B^2| \cdot |S|)$ to $\mathcal{O}(|B| \cdot |S|)$.

Opportunity Cost: Feasibility and Evaluation

Strategies for Viable Solutions and Efficiency

Guaranteeing Feasibility:

- During queue traversal, mark centers as full when activity measure hits the lower bound target: $\sum_{j \in B} X_{ij} v_j^m \geq Y_i \mu_i (1 - t^m), \forall i \in S$
- Ensures even distribution among centers.
- Maintains lower bounds, preventing exceeding upper bounds.
- The location phase produces feasible solutions most of the time.

Partial Evaluation:

- Use additional data structures for activity measures, updating only for specific i and j during allocation.
- For feasibility checks, compute only for the specific center i, adding the activity measure values and risk of j.
- Also utilized in the local search phase.

Opportunity Cost

Pseudocode of Precomputing

```
1: Calculate Opportunity Cost,   $\forall j \in B$, store in priority queue $pq$
2: Compute and Order the Best Assignments,   $\forall j \in B$, store in dictionary $best\_assignments$
3: Initialize $assigned\_clients$ as an empty set
4: Initialize $full\_centers$ as an empty set
5: Initalize $values\_matrix, risk\_vec$ as auxiliary data structures for feasibility state
```

Opportunity Cost

Main Loop

```
1:  Precompute the data structures.
2:  while $pq$ is not empty and $|assigned\_clients| < B$ do
3:      $j \leftarrow$ dequeue $pq$
4:      if $j \notin assigned\_clients$ then
5:          for each $i \in best\_assignments[j]$ do
6:              if $i \notin full\_centers$ and $Y[i] = 1$ then
7:                  $X[i, j] \leftarrow 1$
8:                  Add client to $assigned\_clients$
9:                  Update $values\_matrix$, $risk\_vec$ with $i, j$
10:                 if $i$ has reached lower bounds in activity measures then
11:                     Add $i$ to $full\_centers$
12:                     Break
13:                 end if
14:             end if
15:         end for
16:     end if
17: end while
```

Local Search: Repairing Unfeasible Solutions

Efficient Local Search Strategies

While the opportunity cost matrix doesn't guarantee feasibility, the local search can repair solutions using specific moves:

Add(i)

Add BUs to a center i violating lower bounds until the constraint is met. Ensure removed BUs don't render their previous centers unfeasible.

Remove(i)

Remove BUs from a center i violating upper bounds. Ensure new allocations don't lead to unfeasible centers.

By directing the local search with "Add" or "Remove" sets for centers, we achieve faster computations.

Local Search: Improving Solutions

Variable Neighborhood Descent Moves

With a feasible solution in hand, we implement the following moves:

Allocate_Other(i, j)

Reassign BU j from center i to another center $\tilde{i}$, where $\tilde{i} \neq i$, $\iff X_{ij} = 1$.

Interchange$(i, j, \tilde{i}, \tilde{j})$

Swap allocations of BUs j and $\tilde{j}$ such that j is allocated to $\tilde{i}$ and $\tilde{j}$ to i, $\iff X_{ij} = 1 \wedge X_{\tilde{i}\tilde{j}} = 1$.

Deactivate$(i, \tilde{i})$

Close center i and open previously unused center $\tilde{i}$. Assign BUs to $\tilde{i}$ till lower bounds are met, and reallocate BUs from i to the best-fit centers, $\iff Y_i = 1 \wedge Y_{\tilde{i}} = 0$.

Experiments layout

Computational results

Development Environment:

- *Language*: Julia 1.9.0
- *Platform*: Intel Core i5-9300 2.5 GHz, 16 GB RAM, Windows 10

Source Code:

- FOSS release available at `https://github.com/eduardosalaz/tesis`

Experiment Instances:

- BUs and Centers' coordinates: Randomly between 5 and 10,000
- Incorporated center types, risk, and activity measures generation

Constructive algorithms comparison

Computational results

Data set - Size 1

Instances:	20
Structure:	$\|B\| = 625, \|S\| = 62, p = 32$

Data set - Size 2

Instances:	20
Structure:	$\|B\| = 1250, \|S\| = 155, p = 62$

Exact Solver Comparison

Gurobi Performance

An integral part of our experimental methodology involved solving the mathematical model with an exact solver, Gurobi 9.5.2 with a cutoff time of 1800 seconds.

Results

Out of all the instances, Gurobi only found the optimal solution for one from Dataset 1. For the remaining instances, it couldn't reach the optimal value within the cutoff time.

Constructive algorithms comparison

Computational results

Dataset	Location Strat.	%Feasible		Alloc. Time		Location Time	Total Time		%Rel diff. to Gurobi UB	
		Matrix Alloc.	Queue Alloc.	Matrix Alloc.	Queue Alloc.		Matrix Alloc.	Queue Alloc.	Matrix Alloc.	Queue Alloc.
1	P-disp	0	0	0.38 s	ε	ε	1.84 s	**0.44 s**	41.82	51.87
	SLP	0	0	0.44 s	ε	1 s	3.16 s	1.41 s	48.32	54.91
2	P-disp	0	**90**	3.28 s	ε	ε	11 s	ε	70.12	88.91
	SLP	0	85	3.28 s	ε	7 s	23.73 s	7 s	74.62	79.52

Table: Constructive Heuristics Comparison of Averages.

Total Time includes Repair Solution Phase. For Dataset 1, queue allocation method violates on average 5 constraints, while matrix method violates 46 on average. The *p*-dispersion heuristic was chosen for the Location Phase and Opportunity Cost Queue for Allocation in GRASP.
ε for times faster than 0.1 s

Local Search algorithms comparison

Computational results

Dataset	Strat.	% Rel. Improv.	%Rel diff. to Gurobi UB	Time
1	Best Found	**95.6**	9.95	0.36 s
	First Found	67.12	12.33	**0.27 s**
2	BF	**83.45**	9.85	3.03 s
	FF	71.23	10.85	**1.88 s**

Table: Local Search Heuristics Comparison of Averages.

FF strategy is faster with reasonabe gaps to Gurobi's Upper Bound. The Deactivate move, despite being computationally complex, provides the most significant improvements.

Heuristics results with larger sized instances

Computational results

Components of the Heuristics:

- **Location:** p-dispersion
- **Allocation:** Opportunity cost queue
- **Local Search:** First Found strategy

Data set - Size 3

Instances: 10

Structure: $|B| = 2500, |S| = 325, p = 125$

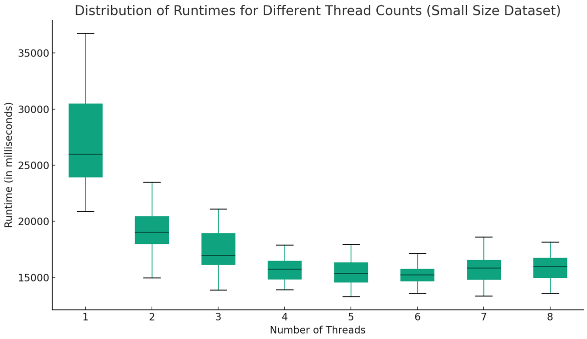
Gurobi's cut-off time was set to 2 hours.

Performance:

- Construction: **0.11 s** + Local search: **14.77 s** = Total Time: **14.87 s**
- Gap to Gurobi's UB: **9.76%**
- **483.2 times (48,320%) faster** than Gurobi.

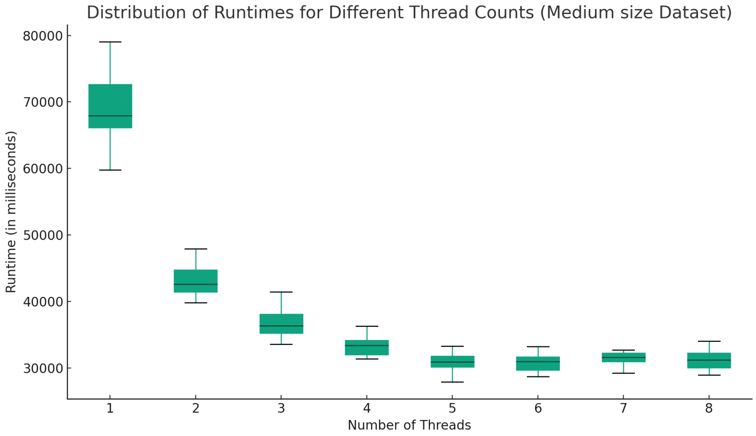
GRASP: Multi-threaded Performance

- Basic GRASP setup: $\alpha_a = 0.3, \alpha_l = 0.3$, $max\_iters = 90$.
- ANOVA showed thread count does not significantly impact objective function value (p-value: 0.9989).



GRASP: Optimal Thread Count

- Paired t-tests complemented the analysis.
- Visual inference suggests optimal thread count is 4 for both datasets.



GRASP: Calibration of Parameters

- We explored calibration of 3 parameters: α_l, α_a, and *max_iters*.
- Tested combinations for subsets of Datasets 1 and 2, totaling 125 configurations:
  - $\alpha_l \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$
  - $\alpha_a \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$
  - $max\_iters \in \{10, 30, 50, 70, 90\}$

GRASP: Findings from Calibration

- Factorial ANOVA and One Way ANOVA:
  - α_l, α_a showed minimal effect on the objective function and runtime.
  - *max_iters* impacted both runtime and objective function.
- Chose $\alpha_l = 0.1$, $\alpha_a = 0.3$ for best mean runtime.
- Increasing *max_iters* from 10 to 90:
  - Improved objective function by 1.541%.
  - Increased runtime by 717.689%.
- Opted for $max\_iters = 70$ for a balance of runtime and solution quality.

Note: The full results table isn't shown due to size constraints.

GRASP Results

Dataset	GRASP Time	Gurobi Time	%Rel diff. to Gurobi UB
1	17.05 s	1800 s	2.88
2	62 s	1800 s	5.23
3	160 s	7200 s	7.91

Table: Comparison of Average values of Fine-tuned GRASP with 4 threads vs Gurobi

Wrap-up

Conclusions

- p-dispersion heuristic for location and opportunity cost with the usage of efficient data structures for allocation offer the best performance in feasibility and speed.
- Fine-tuning GRASP: *max_iters*$= 70$, $\alpha_l = 0.1, \alpha_a = 0.3$ and 4 threads balances runtime and solution quality.
- GRASP's solutions closely rival Gurobi's but are significantly faster. For Dataset 1, GRASP outperformed Gurobi in several instances.

Future work

- Explore risk as an uncertainty measure.
- Introduce contiguity constraints.
- Hybridize with exact formulation.
- Investigate other strategies (TS, ILS, IGLS, VNS, SS).

Acknowledgments

- Thank you very much for your attention!
- Feedback is welcome :)
- Email: eduardosalaz@outlook.com
- GitHub: `https://github.com/eduardosalaz`

References

[1] Jesús Fabián López Pérez, Tahir Ekin, Jesus A. Jimenez, and Francis A. Méndez Mediavilla. "Risk-balanced territory design optimization for a Micro finance institution". In: *Journal of Industrial and Management Optimization* 16.2 (2020), pp. 741–758.

[2] S. S. Ravi and D. J. Rosenkrantz. "Heuristic and Special Case Algorithms for Dispersion Problems". en. In: *Operations Research* 42.2 (1994), pp. 299–310. URL: http://www.jstor.org/stable/171673.

[3] Roger Z. Ríos-Mercado and Hugo Jair Escalante. "GRASP with path relinking for commercial districting". In: *Expert Systems with Applications* 44 (2016), pp. 102–113. ISSN: 0957-4174. URL: https://www.sciencedirect.com/science/article/pii/S0957417415006338.

[4] Roger Z. Ríos-Mercado and Elena Fernández. "A reactive GRASP for a commercial territory design problem with multiple balancing requirements". In: *Computers & Operations Research* 36.3 (2009), pp. 755–776. ISSN: 0305-0548. URL: https://www.sciencedirect.com/science/article/pii/S0305054807002249.