

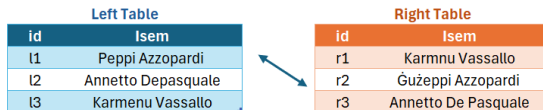
Paper Review - Auto-FuzzyJoin: Auto-Program Fuzzy Similarity Joins Without Labelled Examples

Peng Li, Xiang Cheng, Xu Chu, Yeye He, Surajit Chaudhuri

Carmel Gafa

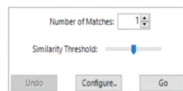
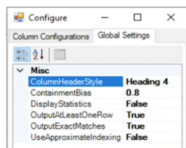
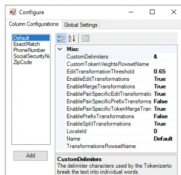
April 5, 2025

Fuzzy-join (or similarity join)



- Fuzzy join takes two tables as inputs and identifies record pairs that refer to the same entity.
- As an example, l1 and r2 refer to the same person.
- The concept can be extended to records with multiple fields or attributes.

Fuzzy-join configuration



- Fuzzy-join has been integrated into many commercial applications
- These systems are often difficult to use due to the large number of configuration parameters.
- The extension in Microsoft Excel has 19 options that span across 3 dialog boxes.
 - 11 are binary, thus resulting in 2048 possible configuration scenarios.
 - 8 continuous, such as thresholds and biases.
- In order to execute quality Fuzzy-joins, these configurations require careful user setup to achieve high-quality results.

Theoretical foundation: fuzzy join mapping

Given a **reference table** L and a table R containing records that may be **imprecise** or noisy, a **fuzzy join mapping** J establishes approximate matches between them.

- J connects elements of R to similar elements in L based on a chosen **similarity measure** (e.g., Levenshtein distance, cosine similarity, Jaccard similarity).
- Each record $r \in R$ is mapped to at most one record $l \in L$, or **no match at all** (denoted by \perp).
- The join is **many-to-one** because multiple records in R can be associated with the **same** record in L , but each $r \in R$ has only **one** possible match.

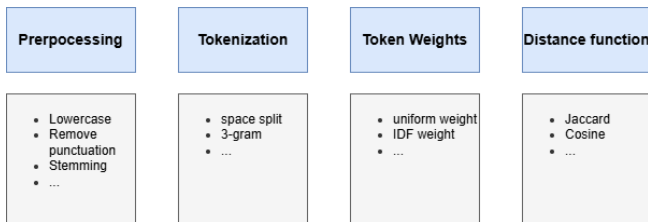
Formally:

$$J : R \rightarrow L \cup \perp$$

Theoretical foundation: fuzzy join configuration space

A **fuzzy join** f compares two strings, r and l , by computing a distance score that reflects their similarity. The computation of this score is governed by a variety of parameters, forming a **parameter space**.

Definition: Each unique combination of these parameters defines a specific **join function** $f \in \mathcal{F}$, where \mathcal{F} is the space of all possible join functions.



Example: fuzzy join distance score computation

Join Function: $f = (L, SP, EW, JD)$

- **L:** Lower-casing (Preprocessing)
- **SP:** Space Tokenization
- **EW:** Equal Weights
- **JD:** Jaccard Distance

Inputs:

- $l = \text{"2012 tigers lsu baseball team"}$
- $r = \text{"2012 lsu baseball team"}$

Tokenization (SP):

- $l \rightarrow \{2012, tigers, lsu, baseball, team\}$
- $r \rightarrow \{2012, lsu, baseball, team\}$

Jaccard Distance:

- $A \cap B = \{2012, lsu, baseball, team\} \rightarrow |A \cap B| = 4$
- $A \cup B = \{2012, tigers, lsu, baseball, team\} \rightarrow |A \cup B| = 5$
- Jaccard Similarity $= \frac{4}{5} = 0.8$
- Jaccard Distance $= 1 - 0.8 = 0.2$

Result: $f(l, r) = 0.2$

Theoretical foundation: threshold and join configuration

- Once the distance $f(l, r)$ is computed:
 - It is compared to a threshold **compared to a threshold** θ to decide whether to join the string pair l and r .
 - lower θ gives stricter matches
 - If $f(l, r) \leq \theta$, the pair is considered a **match**.
- Together, the function f and the threshold θ define what the authors call a **join configuration**:

$$C = \langle f, \theta \rangle$$

- This configuration encapsulates both:
 - How distance is computed.
 - When two strings are considered similar enough to be joined.

Definition: A join configuration C is a 2-tuple $C = \langle f, \theta \rangle$, where $f \in \mathcal{F}$ is a join function, and θ is a threshold.

We use $\mathcal{S} = \{ \langle f, \theta \rangle \mid f \in \mathcal{F}, \theta \in \mathbb{R} \}$ to denote the space of join configurations.

Theoretical foundation: fuzzy join mapping

Given two tables L and R , a join configuration $C \in \mathcal{S}$ induces a **fuzzy join mapping** J_C , defined as:

$$J_C(r) = \arg \min_{l \in L, f(l,r) \leq \theta} f(l, r), \forall r \in R$$

That is

- For each record $r \in R$, find $l \in L$ that minimizes the distance $f(l, r)$, **only if** that distance is less than or equal to the threshold θ .
- If no such $l \in L$ exists such that $f(l, r) \leq \theta$, then $J_C(r)$ maps to \perp — i.e., no match for that record.

Theoretical foundation: the problem with single join configurations

Real-world data can exhibit **multiple types of variations simultaneously**, such as:

- **Typos**
- **Missing tokens**
- **Extraneous information**

As a result, relying on a **single join configuration** often fails to capture all valid matches, particularly when high **recall** is required.

To handle this diversity, the algorithm uses a **set of join configurations**:

$$U = \{C_1, C_2, \dots, C_K\}$$

Instead of relying on a single configuration, the system computes join results from each one.

This approach allows the system to:

- Accommodate diverse types of variations.
- Improve overall recall by **combining multiple perspectives** on similarity (different parametrizations that are sensitive to different types of noise).

L-id	L-Table (Reference Table)		R-id	R-Table (Input Table)
l_1	2008 LSU Tigers baseball team	✓	r_1	2008 LSU baseball team
l_2	2008 LSU Tigers football team	✓	r_2	2008 LSU football team
l_3	2008 Mississippi State Bulldogs baseball team	✓	r_3	2008 Mississippi State Bulldog baseball team
l_4	2008 Mississippi State Bulldogs football team	✓	r_4	2008 Mississippi State Bulldog football team
l_5	...		r_5	...
l_6	2007 LSU Tigers football team	✗	r_6	2007 LSU Tigers baseball team
l_7	2007 Wisconsin Badgers football team	✗	r_7	2008 Wisconsin Badgers football team
l_8	2011 LSU Tigers football team	✗	r_8	2010 LSU Tigers football team
l_9	2007 LSU Tigers baseball team	✗	r_9	2007 LSU Tigers football team

- A **Jaccard distance** with threshold 0.2 works well for pairs like (l_1, r_1) , which differ by only one or two tokens.
- However, for pairs like (l_3, r_3) with **spelling variations**, Jaccard similarity is not enough:
 - Jaccard distance $\approx 0.5 \rightarrow$ too high to match under the 0.2 threshold
 - A more suitable metric is **Edit Distance**, which can better align such pairs.

Theoretical foundation: fuzzy join via multiple configurations

- To handle this diversity, the algorithm uses a **set of join configurations**:

$$U = \{C_1, C_2, \dots, C_K\}$$

- Instead of relying on a single configuration, the system computes join results from each.
- This approach allows the system to:
 - Accommodate diverse types of variations.
 - Improve overall recall by **combining multiple perspectives** on similarity.

Given L and R , a set of join configurations $U = \{C_1, C_2, \dots, C_K\}$ induces a **fuzzy join mapping** J_U , defined as:

$$J_U(r) = \bigcup_{C_i \in U} J_{C_i}(r), \forall r \in R$$

This means that the overall result of the fuzzy join using configuration set U is the **union** of results from all individual configurations $C_i \in U$.

Each configuration $C_i \in U$ is designed to capture a **specific type of string variation** (e.g., typos, missing tokens, extra tokens).

Two records are considered **joined by the set U** if and only if they are joined by **at least one** configuration $C_i \in U$.

- Each configuration contributes **high-quality joins** targeted at particular data challenges.
- The overall join is more **robust and comprehensive**.

Theoretical foundation: evaluating join quality; Precision

Given two tables R and L , and a **space of join configurations** S , the objective is to find a subset $U \subseteq S$ that produces **good fuzzy join results**.

Let:

- J_U be the fuzzy join mapping induced by configuration set U
- J_G be the **ground truth** join mapping — the ideal join result

Precision measures how many of the predicted joins are correct:

$$\text{precision}(U) = \frac{\underbrace{|\{r \in R \mid J_U(r) \neq \emptyset, J_U(r) = J_G(r)\}|}_{\text{True Positives (TP)}}}{\underbrace{|\{r \in R \mid J_U(r) \neq \emptyset\}|}_{\text{TP + FP (all predicted joins)}}}$$

- **Numerator (TP)**: Records where a join was predicted and it matched the ground truth.
- **Denominator (TP + FP)**: All records where a join was predicted (correct or not).
- Only records with a prediction (i.e., $J_U(r) \neq \emptyset$) are evaluated in this precision formula.

Theoretical foundation: evaluating join quality; Recall

Recall measures how many of the correct (ground truth) joins were successfully predicted:

$$\text{recall}(U) = \underbrace{|\{r \in R \mid J_U(r) \neq \emptyset, J_U(r) = J_G(r)\}|}_{\text{True Positives (TP)}}$$

- This is the **absolute count of True Positives**, i.e., records for which:
 - A join was predicted ($J_U(r) \neq \emptyset$), and
 - It matches the ground truth ($J_U(r) = J_G(r)$)

False Negatives (FN) — cases where a correct join was missed — are defined as:

$$\text{FN} = |\{r \in R \mid J_G(r) \neq \emptyset, J_U(r) = \emptyset\}|$$

Note: The denominator $TP + FN$ is constant across all U for a fixed dataset, so it is omitted in comparisons.

Theoretical foundation: estimating Precision/Recall for a single join configuration

Given:

- A **single join configuration** $C = \langle f, \theta \rangle$
- Two tables:
 - L : reference table
 - R : query table

Assumption: Complete Reference Table L

- L is assumed to contain **all possible true matches** for records in R .
- Ensures that for each $r \in R$, there exists a correct match $l \in L$.
- Simplifies analysis by reducing the chance of missing true positives due to an incomplete reference.

Geometric View of the Distance Function f

- Join function f embeds records into a **metric space**.
- Records are conceptually modelled as points on a **unit grid**.
- Each $l \in L$ is surrounded by **close variants** (differing by a token, character, etc.).
- The distance between each l and the surrounding r 's is exploited by θ to compute join pairs.

Analogy: Stars and Planets

- Reference records $l \in L$ are like **stars** on a grid.
- Query records $r \in R$ are like **planets** that orbit these stars.
- Identifying the best join $J_C(r)$ is like determining **which star a planet orbits**.

Theoretical foundation: safe joins and the geometry of fuzzy matching

Safe Joins with a Complete L

- Define the **grid width** w : typical distance between a record l and its closest neighbors in L .
- A join is considered **safe** if the distance $d = f(l, r)$ satisfies:

$$d < \frac{w}{2}$$

- This guarantees that r lies **closer to its true match** l than to any other reference point.

Why This Matters:

- Ensures high **precision** — avoiding false positives caused by ambiguous joins.
- Avoids joining r to an incorrect l' that lies at a similar distance.

Analogy: Stars and Planets

- A planet that lies **equidistant** between two stars (at $\frac{w}{2}$ each) **cannot be confidently claimed by either**.
- In fuzzy joining, such cases are inherently **ambiguous** and risky to resolve.

Theoretical foundation: estimating join precision (local heuristic)

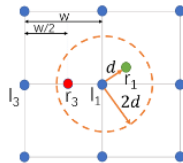
Given a query record $r \in R$ and its closest match $l \in L$, with distance $d = f(l, r)$, we can estimate how **precise** this join is — i.e., how likely it is that (l, r) is a **correct match**.

- The **more candidate records** in L that are close to r , the **less confident** we are about any one being the true match.
- So we count how many other records $l' \in L$ fall within the **$2d$ -ball** centered at l :

$$\text{precision}(l, r) = \frac{1}{\underbrace{|\{l' \in L \mid f(l, l') \leq 2f(l, r)\}|}_{\text{TP} + \text{FP (local competitors)}}}$$

- A small $2d$ -ball \rightarrow high precision (few competitors).
- A large $2d$ -ball \rightarrow low precision (many competitors).

This provides a **data-driven estimate** of join quality **without needing ground truth**.



- To estimate the quality of joining r_1 , we first find its nearest neighbor in L , which we'll call l_1 .
- Compute the distance: $d = f(l_1, r_1)$.
- Draw a ball of radius $2d$ centered at l_1 .
 - If no other L records fall in the ball \rightarrow high confidence.
- In this case, the $2d$ -ball contains only l_1 :

$$\text{precision}(l_1, r_1) = \frac{1}{1} = 1$$

- **High confidence join.**

Theoretical foundation: When L is incomplete

Problem: When L is incomplete (i.e., some records are missing):

- Missing records in L result in **missing stars** in the grid.
- A record r may join to the wrong l , causing **false positives** and reducing **precision**.
- Example: If r_2 should match with l_2 (but l_2 is missing), it might instead match l_1 using $d = f(r_2, l_1)$.

Note:

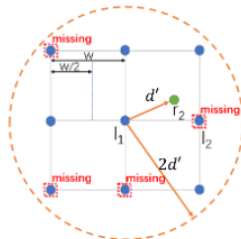
- Even if some records in L are missing, **safe decisions** can still be made.

Precision estimation:

- r_2 should match l_2 (missing), so l_1 becomes the fallback.
- The $2d'$ -ball around l_1 contains **5 records**.
- Precision:

$$\text{precision}(l_1, r_2) = \frac{1}{5}$$

- \Rightarrow **Low confidence join**



- r_2 should join with l_2 , but l_2 is **missing**
- l_1 becomes the closest available record
- Compute distance $d' = f(l_1, r_2)$
- Draw a $2d'$ -ball around l_1
 - If the ball includes many other L records $\rightarrow d'$ is too **lax**
 - Join becomes unreliable

Theoretical foundation: estimating Precision and Recall for a configuration

A configuration $C = \langle f, \theta \rangle$ includes:

- A join function f
- A threshold θ

1. Local precision for a join

$$\text{precision}(r, C) = \frac{1}{|\{l' \in L \mid f(l, l') \leq 2f(l, r)\}|}$$

- $J_C(r) = l$: join match for $r \in R$
- Denominator = number of plausible alternatives

2. Expected true positives

$$TP(C) = \sum_{r \in R, J_C(r) \neq \emptyset} \text{precision}(r, C)$$

3. Expected false positives

$$FP(C) = \sum_{r \in R, J_C(r) \neq \emptyset} (1 - \text{precision}(r, C))$$

4. Overall Precision and Recall

$$\text{precision}(C) = \frac{TP(C)}{TP(C) + FP(C)} \quad \text{recall}(C) = \frac{TP(C)}{TP(C)}$$

Note: Recall is estimated absolutely since ground truth is unavailable.

Example: Precision and Recall estimation

Setup: Assume 3 records in R , joined to L using configuration $C = \langle f, \theta \rangle$.

Join Results:

- $J_C(r_1) = l_1, f(l_1, r_1) = 0.1$, 5 plausible matches $\Rightarrow \text{precision}(r_1, C) = \frac{1}{5} = 0.20$
- $J_C(r_2) = l_2, f(l_2, r_2) = 0.05$, 2 plausible matches $\Rightarrow \text{precision}(r_2, C) = \frac{1}{2} = 0.50$
- $J_C(r_3) = l_3, f(l_3, r_3) = 0.2$, 4 plausible matches $\Rightarrow \text{precision}(r_3, C) = \frac{1}{4} = 0.25$

Estimated TP and FP:

$$TP(C) = 0.20 + 0.50 + 0.25 = 0.95$$

$$FP(C) = (1 - 0.20) + (1 - 0.50) + (1 - 0.25) = 2.05$$

Estimated Precision and Recall:

$$\text{precision}(C) = \frac{0.95}{0.95 + 2.05} = \frac{0.95}{3.00} \approx \mathbf{0.317}$$

$$\text{recall}(C) = TP(C) = \mathbf{0.95}$$

Note: This example assumes no ground truth; hence recall is based on expected TP count.

Theoretical foundation: Precision and Recall for a set of configurations

Let $U = \{C_1, C_2, \dots, C_K\}$ be a set of configurations.

Case 1: No Conflicts in U

- Each record $r \in R$ is matched by at most one configuration:

$$\forall r \in R, \quad |J_U(r)| \leq 1$$

- Then:

$$TP(U) = \sum_{C \in U} TP(C), \quad FP(U) = \sum_{C \in U} FP(C)$$

Case 2: Conflicting Assignments in U

- Multiple configurations suggest different joins for the same r
- Resolve conflicts by:
 - Compare precision scores: $\text{precision}(r, C_i)$ vs. $\text{precision}(r, C_j)$
 - Choose the match with higher precision
 - Assign that join to $J_U(r)$
 - Recompute $TP(U)$ and $FP(U)$

Final Estimates:

$$\text{precision}(U) = \frac{TP(U)}{TP(U) + FP(U)} \quad \text{recall}(U) = \frac{TP(U)}{TP(U) + FP(U)}$$

Example: resolving conflicting joins from multiple configurations

Context: Two configurations propose different joins for the same record $r \in R$ using different string similarity methods.

Configurations:

- $C_1 = \langle f_1, \theta_1 \rangle$, where f_1 uses Jaccard distance over space-tokenized lowercase strings with equal weights.
- $C_2 = \langle f_2, \theta_2 \rangle$, where f_2 uses Cosine similarity over character trigrams with TF-IDF weighting.

Join Proposals for r :

- C_1 : $J_{C_1}(r) = l_1$ with $\text{precision}(r, C_1) = \frac{1}{4} = 0.25$
- C_2 : $J_{C_2}(r) = l_2$ with $\text{precision}(r, C_2) = \frac{1}{2} = 0.50$

Conflict Resolution Strategy:

- 1 Compare estimated precision:

$$\text{precision}(r, C_1) = 0.25 < \text{precision}(r, C_2) = 0.50$$

- 2 Assign $J_U(r) = l_2$ (higher-confidence match from C_2)

Effect:

- $TP(U)$ and $FP(U)$ incorporate only the winning match.
- Competing matches are discarded.

Auto-FuzzyJoin Algorithm: single column case

Recall-Maximizing Fuzzy Join (RM-FJ) is **NP-hard**. Use a **greedy approximation algorithm** called AutoFJ.

Objective:

- Maximize recall $TP(U)$ subject to maintaining precision $(U) \geq \tau$

Greedy Strategy:

- Select configurations that:
 - Increase true positives (recall)
 - Minimize false positives (preserve precision)
- Guided by the **Profit Metric**:

$$\text{profit}(U) = \frac{TP(U)}{FP(U)}$$

Blocking Heuristic:

- To reduce the number of comparisons, apply **3-gram blocking**.
- Each string is decomposed into overlapping sequences of 3 characters (3-grams).
- Only record pairs that share at least one common 3-gram are considered for joining.
- This blocks out obviously dissimilar pairs and speeds up computation.
- Applied to both $L-L$ and $L-R$ candidate pairs:

$LL, LR \leftarrow$ generate candidate pairs using 3-gram overlap

Algorithm 1 AUTOFJ for single column

Require: Tables L and R , precision target τ , search space S

```
1:  $LL, LR \leftarrow$  apply blocking with  $L - L$  and  $L - R$ 
2:  $LR \leftarrow$  Learn negative-rules from  $LL$  and apply rules on  $LR$  (Alg. 2)
3: Compute distance with different join functions  $f \in S$ 
4: Pre-compute precision estimation for each configuration  $C \in S$ 
5:  $U \leftarrow \emptyset$ 
6: while  $S \setminus U \neq \emptyset$  do
7:    $max\_profit \leftarrow 0$ 
8:   for all  $C \in S \setminus U$  do
9:     if  $profit(U \cup \{C\}) > max\_profit$  then
10:       $C^* \leftarrow C, max\_profit \leftarrow profit(U \cup \{C\})$ 
11:   if  $precision(U \cup \{C^*\}) > \tau$  then
12:      $U \leftarrow U \cup \{C^*\}$ 
13:   else
14:     break
15: return  $U$ 
```

1. 3-Gram blocking using TF-IDF

LL, LR \leftarrow apply 3-gram blocking on $L-L$ and $L-R$

Reference Table L :

l_1	"john smith"
l_2	"jane smythe"
l_3	"alice johnson"

Query Record r_1 : "jon smyth"

Step 1: Preprocessing (P)

- Lowercasing (already lowercase)
- Add padding for 3-grams: e.g., "john smith" \rightarrow "##john#smith##"

Step 2: Tokenization (T)

- r_1 : ##j, #jo, jon, on# , n#s, #sm, smy, myt, yth, th#, h##
- l_1, l_2 : similar 3-gram sequences

Step 3: Token Weighting (W)

- Use TF-IDF to emphasize rare, meaningful trigrams (e.g., smy, yth)
- r_1 - l_2 : **High score(rare overlapping trigrams)**, r_1 - l_1 : Medium (more common overlap), r_1 - l_3 : Zero (no shared trigrams)

Blocking Result:

- Only compare r_1 with $l_1, l_2 \rightarrow$ prune l_3

2. Optimization - filtering with negative rules

$LR \leftarrow$ Learn negative-rules from LL and apply rules on LR (Alg. 2)

Assumption: Although 3-gram blocking may have pruned l_3 , we assume here it was retained due to weak overlap, allowing us to illustrate negative-rule filtering.

Goal: Use **obvious non-matches** in $L-L$ to learn rules that help **filter unlikely $L-R$ pairs** before costly distance computations.

Step 1:	Generate	LL	—	Self-Join	on	L	using	3-gram	blocking
Pair	Shared 3-grams	Interpretation							
l_1 vs l_2	sm, smy, th	Possibly similar							
l_1 vs l_3	jo, on	Clearly different							
l_2 vs l_3	Weak overlap	Probably different							

Learn Negative Rule:

"If 3-gram overlap ≤ 2 , treat as a non-match."

Step 2: Apply Rule on LR Candidate Pairs	Pair	Overlap	Apply Rule?	Keep?
	r_1, l_1	~ 4	No	Yes
	r_1, l_2	~ 5	No	Yes
	r_1, l_3	~ 1	Yes	No

Effect: Filter out clearly irrelevant pairs early — no need to compute Jaccard or Edit Distance!

3. Compute distances - apply join functions

Compute distance with different join functions $f \in \mathcal{S}$

Once candidate pairs are identified (via blocking and optional negative rules), we compute the actual similarity using multiple join functions $f \in \mathcal{S}$.

Each join function is defined by:

- Preprocessing (e.g., lowercasing, punctuation removal)
- Tokenization (e.g., char 3-grams, word tokens)
- Token weights (e.g., TF-IDF)
- Distance function (e.g., Jaccard, Cosine, Edit)

	r (query)	l (reference)
Example Candidate Pairs (after blocking):	"jon smyth"	"john smith"
	"jon smyth"	"jane smythe"

	Function f	Tokenizer	Distance	Description
Join Functions in \mathcal{S} :	f_1	char 3-grams	Jaccard	Overlap in token sets
	f_2	char 3-grams	Cosine (TF-IDF)	Weighted similarity
	f_3	raw string	Levenshtein	Edit distance
Computed Scores:	Pair	f_1	f_2	f_3
	jon vs john	0.4	0.5	2
	jon vs jane	0.6	0.7	3

Note: Distances may follow different scales — lower often means more similar.

Start of Greedy Algorithm

Initialize: $U \leftarrow \emptyset$

U will hold the **selected join configurations**:

$$C = \langle f, \theta \rangle$$

Each configuration includes:

- A join function $f \in \mathcal{F}$ (defined by P, T, W, D)
- A distance threshold θ (max allowed distance for a match)

Goal:

- Select a subset $U \subseteq \mathcal{S}$ from all candidate configurations
- Maximize recall: $TP(U)$
- Maintain precision: $\text{precision}(U) \geq \tau$

Example: Precomputed Configuration Set \mathcal{S}

Config C	Description
$C_1 = \langle f_1, 0.37 \rangle$	Jaccard distance with $\theta = 0.37$
$C_2 = \langle f_2, 0.42 \rangle$	Cosine distance with $\theta = 0.42$
$C_3 = \langle f_3, 2 \rangle$	Edit distance with $\theta = 2$

These θ values were selected based on prior precision–recall evaluation for each f .

Main Greedy Loop

Main Loop: while $S \setminus U \neq \emptyset$ do

We continue as long as there are still unused configurations to consider.

Notation:

- S : full set of candidate configurations, each $C = \langle f, \theta \rangle$
- U : set of selected configurations
- $S \setminus U$: unused configurations

At each iteration:

- 1 Evaluate each $C \in S \setminus U$
- 2 Compute profit: how many true positives vs. false positives it contributes
- 3 Select the best configuration C^*
- 4 If $\text{precision}(U \cup \{C^*\}) \geq \tau$:

$$U \leftarrow U \cup \{C^*\}$$

Example State:

- $S = \{\langle f_1, 0.37 \rangle, \langle f_2, 0.42 \rangle, \langle f_3, 2 \rangle\}$
- $U = \emptyset$

Loop continues while there are remaining candidates and precision can be preserved.

Find Most Promising Configuration (Profit Heuristic)

Pseudocode:

```
max_profit ← 0
for all  $C \in S \setminus U$  do
    if  $\text{profit}(U \cup \{C\}) > \text{max\_profit}$  then
         $C^* \leftarrow C, \text{max\_profit} \leftarrow \text{profit}(U \cup \{C\})$ 
```

Profit Formula:

$$\text{profit}(U \cup \{C\}) = \frac{TP(U \cup \{C\})}{FP(U \cup \{C\})}$$

	Config C	TP	FP	Profit = TP / FP
Example:	C_1	4	2	2.0
	C_2	5	5	1.0
	C_3	3	1	3.0

After evaluation: $C^* = C_3, \text{max_profit} = 3.0$

Heuristic: choose the configuration that gives the most recall "bang" per unit of precision "risk."

Precision Constraint Check & Termination

Check: if $\text{precision}(U \cup \{C^*\}) > \tau$ then $U \leftarrow U \cup \{C^*\}$

After selecting the best candidate C^* (based on profit), we must verify that adding it to U preserves minimum required precision τ .

- If precision passes: add C^* to U
- Else: break — no remaining configs will satisfy the constraint

Example 1 (Pass):

Config	TP	FP	Profit	Precision	τ
C_3	3	1	3.0	0.75	0.7

$\Rightarrow \text{Precision} > \tau \rightarrow \text{Accept} \rightarrow U \leftarrow \{C_3\}$

Example 2 (Fail & Break):

Config	TP	FP	Profit	Precision	τ
C_3	3	1	3.0	0.75	0.8

$\Rightarrow \text{Precision} < \tau \rightarrow \text{Reject} \rightarrow \text{Stop Loop}$

Greedy termination: If best config can't meet τ , no others will.

Exit: Return Final Join Plan

Return: U

The greedy loop terminates when:

- $S \setminus U = \emptyset$ (all configs evaluated), or
- The best candidate fails the precision constraint

The algorithm returns U : a set of selected configurations:

- Each $C = \langle f, \theta \rangle$
- Maximizes recall while keeping $\text{precision}(U) \geq \tau$

Each configuration in U defines:

- A join function f (e.g., Jaccard, Cosine, Edit Distance)
- A threshold θ used to accept matches

Example Output:

- $U = \{ \langle f_2 = \text{Cosine}, \theta = 0.5 \rangle, \langle f_3 = \text{Edit}, \theta = 2 \rangle \}$

These are used to perform the final fuzzy similarity join.

Selecting the Best Match: Full Example

1. Input Setup:

- Query record: $r_1 = \text{"jon smyth"}$
- Reference table: $L = \{\text{"john smith"}, \text{"jane smythe"}, \text{"alice johnson"}\}$
- After blocking: candidates for r_1 are l_1 and l_2

	Join Function f	θ	$f(r_1, l_1)$	$f(r_1, l_2)$	Matches?
2. Distance Results:	Jaccard (3-grams)	0.4	0.5	0.3	l_2 only
	Cosine (TF-IDF)	0.5	0.6	0.4	l_2 only
	Edit Distance	2.0	2	3	l_1 only

3. Final Configuration Set U :

- $U = \{\langle f_2 = \text{Cosine}, \theta = 0.5 \rangle, \langle f_3 = \text{Edit}, \theta = 2 \rangle\}$
- Under Cosine: $r_1 \mapsto l_2$
- Under Edit Distance: $r_1 \mapsto l_1$

4. Conflict Resolution: Local Precision

$$\text{precision}(r, C) = \frac{1}{|\{l' \in L \mid f(l, l') \leq 2f(l, r)\}|}$$

Config C	Match	$f(l, r)$	$2d$ -ball size	Precision
C_2 (Cosine)	l_2	0.4	5	$1/5 = 0.2$
C_3 (Edit)	l_1	2	2	$1/2 = 0.5$

Result: *"jon smyth" is matched to "john smith" (Edit Distance), since it has higher estimated precision ($0.5 > 0.2$).*