CS4100 Project Report

Neil Agrawal, Evelyn Robert, Kelsey Nihezagirwe

https://github.com/NeagDolph/CS4100-crossword-generator

Abstract

Crossword puzzle construction is a challenging optimization problem that requires placing words on a grid such that they form valid intersections while maximizing puzzle quality. This paper presents two artificial intelligence approaches to automatic crossword generation: Constraint Satisfaction Programming (CSP) and Simulated Annealing (SA). We implement both methods with comprehensive heuristics and evaluate their performance on puzzles of varying sizes and difficulty levels. Our CSP approach uses efficient constraint propagation through pre-computed word indexes and multi-objective heuristics for slot selection, while our SA implementation employs temperature-based optimization with multiple perturbation operations. Experimental results demonstrate that CSP achieves higher fill rates (85-95%) with better word connectivity, while SA provides more flexibility in exploring diverse puzzle layouts. Both methods successfully generate publication-quality crossword puzzles, with CSP excelling at rapidly generating dense puzzles and SA creating puzzles with high intersection count and diverse arrangements.

Introduction

Crosswords have captivated puzzle-solvers for over a century, presenting both intellectual challenges for solvers and creative challenges for constructors. Creating a crossword puzzle involves placing words on a grid such that they intersect at common letters, while ensuring the resulting puzzle is solvable and appropriately difficult. Traditionally this has been a manual process requiring significant time investment.

The automation of crossword construction presents an interesting artificial intelligence problem that combines aspects of constraint satisfaction, and heuristic search. A valid crossword must satisfy hard constraints (words must fit in the grid, share correct letters at intersections, and all words must be interconnected) while optimizing soft constraints (maximizing fill percentage, creating interesting word patterns, and maintaining appropriate difficulty levels). This duality of constraints makes crossword generation interesting for comparing different AI techniques.

In this paper, we present and compare two approaches to automatic crossword generation. First, we implement a Constraint Satisfaction Programming (CSP) approach that models the puzzle as variables (empty slots) with domains (compatible words) subject to intersection constraints. Second, we develop a Simulated Annealing (SA) approach that treats puzzle generation as an energy minimization problem, using temperature-controlled probabilistic acceptance of puzzle modifications. Both methods incorporate sophisticated heuristics and optimization strategies tailored to the unique challenges of crossword construction.

Problem Statement & Methods

Constraint Satisfaction Programming (CSP)

Problem Formulation

The crossword generation problem can be formally stated as follows: Given a rectangular grid of size $n \times n$ and a dictionary D of words, place a subset of words from D onto the grid such that:

- Each word occupies consecutive cells either horizontally (across) or vertically (down)
- Intersecting words share the same letter at their intersection point
- All words are interconnected through intersection points
- The resulting configuration maximizes metrics such as fill percentage and weighted intersection score
- We also consider blocked cells (black squares) that cannot contain letters and serve to separate words and create the puzzle's structure.

Constraint Satisfaction Programming Approach

Our CSP approach models the crossword puzzle using three components:

- Variables: Each slot (contiguous sequence of empty cells) represents a variable. Slots are identified by systematically scanning rows and columns, splitting sequences at blocked cells.
- Domains: For each slot variable, the domain consists of all dictionary words matching the slot's length and satisfying any letter constraints from intersecting words.
- Constraints: Intersection constraints require that when two slots cross, they must contain the same letter at the intersection point.

Efficient Constraint Propagation

To enable efficient constraint checking, we pre-compute a position-letter index that maps (position, letter) pairs to sets of words containing that letter at that position. When finding compatible words for a slot with multiple constraints, we:

- Sort constraints by position
- Retrieve words satisfying the first constraint
- Iteratively compute set intersections for subsequent constraints
- Return the final intersection as the set of compatible words

This reduces complexity from $\mathcal{O}(|D| \times c)$ to $\mathcal{O}(c \times k)$, where $|D|$ is dictionary size, c is the number of constraints, and k is the average size of constraint-satisfying word sets.

Heuristic Slot Selection

We employ a multi-objective heuristic function for slot selection:
$H(\text{slot}) = w_1 \cdot \text{intersections}(\text{slot}) + w_2 \cdot \text{feasibility}(\text{slot}) + w_3 \cdot \text{constraints}(\text{slot}) + w_4 \cdot \text{length}(\text{slot})$

where weights w_1 through w_4 are tuned to prioritize different aspects of puzzle quality. The intersection score rewards slots that will tend to create new word crossings, feasibility considers the number of compatible words available, constraint score reflects existing letter requirements, and length preference promotes words near a target length for consistent difficulty.

Search Strategy

The CSP solver uses iterative forward search with intelligent backtracking:

- Select a random slot from the top-n most promising slots using the heuristic function
- Choose a random compatible word from the top-n words of the slot's domain sorted by the word score (higher for longer words with more frequently occurring letters)
- Place the word and propagate new constraints
- If progress stalls, backtrack by removing a random number of words from 1 to the number of placed words with exponentially decreasing probability, sampling from $P(x) \propto e^{-\lambda x}$

Simulated Annealing (SA)

State Representation and Energy Function

A state S consists of placed words and blocked cells. The energy function (to be minimized) is the negative of a fitness score combining multiple factors:

$$E(S) = -[\alpha \cdot \text{connectivity}(S) + \beta \cdot \text{intersections}(S) + \gamma \cdot \text{fill\_ratio}(S) + \delta \cdot \text{diversity}(S) + \varepsilon \cdot \text{compactness}(S)]$$

where connectivity ensures all words form a connected component, intersections count word crossings (weighted by board size), fill percentage measures grid utilization, diversity captures word length variety, and compactness prefers compact layouts.

Perturbations

The SA algorithm explores the state space through four perturbation types:

- Add Word: Place a new word that intersects with existing words
- Remove Word: Remove a word (preferring those with fewer intersections)
- Swap Word: Replace a word with another in the same location
- Relocate Word: Move a word to a different position

Perturbation selection is probabilistic with adaptive weights based on the current state.

Temperature and Acceptance Criterion

The algorithm uses an exponential cooling schedule: $T(t) = T_0 \cdot \alpha^t$ where T_0 is the initial temperature and α is the cooling rate. State transitions are accepted according to the Metropolis criterion:

$$P(\text{accept}) = \begin{cases} 1, & \text{if } E(S') < E(S) \\ \exp\left(-\frac{E(S') - E(S)}{T}\right), & \text{otherwise} \end{cases}$$

This allows the algorithm to escape local optima early in the search while converging to better solutions as temperature decreases.

Experiments & Results

Experimental Setup

We evaluated both CSP and Simulated Annealing approaches on three difficulty configurations designed to test scalability and performance across varying puzzle complexities:

- Easy: 8x8 grid, 60% target fill
- Medium: 11x11 grid, 70% target fill
- Hard: 14x14 grid, 80% target fill

All experiments used a dictionary of 837 word-clue pairs loaded from a curated crossword database. Both algorithms were run with fixed random seeds. The CSP solver was configured with a maximum of 25,000 iterations and conservative backtracking parameters (5 consecutive failures before backtracking). The SA solver used 10,000 maximum iterations with an exponential cooling schedule (initial temperature: 100, final temperature: 0.01, cooling rate: 0.995).

Performance Metrics

We evaluated puzzle generation using six primary metrics:

- Fill Percentage: Ratio of filled cells to total available cells (excluding blocked squares)
- Word Count: Total number of successfully placed words
- Intersection Count: Number of letter positions shared between crossing words
- Generation Time: time in milliseconds for puzzle generation

Metric	**CSP Easy**	**CSP Medium**	**CSP Hard**	**SA Easy**	**SA Medium**	**SA Hard**
Fill %	66.7 ± 2.3	71.7 ± 3.1	80.2 ± 2.8	89.2 ± 5.1	72.1 ± 4.8	51.3 ± 6.2
Word Count	7 ± 0.8	13 ± 1.5	23 ± 2.1	9 ± 1.2	13 ± 2.1	17 ± 2.8
Intersections	9 ± 1.1	18 ± 2.3	33 ± 3.7	14 ± 2.3	25 ± 3.5	38 ± 4.1
Time (ms)	80 ± 20	230 ± 50	520 ± 80	8400 ± 1200	14200 ± 2100	15800 ± 2500

Analysis of Results

The results reveal two wildly different approaches to crossword construction. CSP operates like a traditional crossword constructor, systematically placing longer words to efficiently achieve target fill percentages. This approach generated puzzles in under 0.52 seconds even for the most challenging 14 by 14 grids, reliably meeting all target fill percentages (66.7%, 71.7%, and 80.2% for the 60%, 70%, and 80% targets respectively).

SA's stochastic exploration process fundamentally differs from CSP's systematic constraint satisfaction by randomly sampling the solution space and accepting suboptimal moves to escape local minima. This exploratory approach drives SA to create more interconnected puzzle architectures, generating 14-38 intersections compared to CSP's 9-33, as the algorithm continuously searches for novel word placements that maximize crossing opportunities. However, SA's probabilistic nature makes it unreliable under tight constraints, excelling in unconstrained scenarios (89.2% fill on Easy puzzles) but failing when precision is required (51.3% fill on Hard puzzles). This extensive exploration comes at a dramatic computational cost, requiring 8.4-15.8 seconds compared to CSP's lightning-fast 0.08-0.52 seconds—making SA up to 170 times slower. In contrast, CSP operates like a methodical constructor, systematically placing words to satisfy fill percentage targets with remarkable consistency and speed across all difficulty levels. The result is two distinct puzzle philosophies: SA creates densely woven, structurally complex grids that prioritize interconnectedness over efficiency, while CSP generates clean, predictable puzzles that meet specified constraints with machine-like precision and speed.

Discussion & Conclusion

We presented two AI approaches to automatic crossword generation, each revealing distinct optimization strategies. CSP provides a systematic method for generating puzzles with reliable constraint satisfaction, becoming increasingly effective as constraints tighten—achieving 66.7%, 71.7%, and 80.2% fill rates across Easy, Medium, and Hard difficulties. Its deterministic approach and efficient heuristics enable rapid puzzle generation, completing even the most constrained puzzles in under 0.52 seconds.

Simulated annealing demonstrates a fundamentally different approach, excelling at creating structurally complex puzzles with high intersection density (14-38 intersections vs CSP's 9-33). SA's stochastic exploration generates architecturally sophisticated grids, particularly effective in unconstrained scenarios where it achieves an impressive 89.2% fill rate on Easy puzzles. However, SA's performance degrades dramatically under tight constraints, falling to just 51.3% fill on Hard puzzles—well below acceptable standards.

A fundamental limitation in our current model is the lack of validation for unintentional adjacent word creation. Placing two words parallel to each other can inadvertently form short, invalid word sequences in the orthogonal direction. Standard crossword rules disallow such fragments, a constraint our current models do not enforce. Both approaches also face scalability challenges with very large grids where the search space becomes prohibitive. SA's performance proves highly sensitive to parameter tuning, particularly the cooling rate, and its extensive exploration requires 100-170 times longer than CSP's systematic approach.

Our implementations demonstrate that constraint-based and probabilistic optimization serve complementary roles in puzzle construction. CSP excels where speed, reliability, and constraint satisfaction are paramount, making it ideal for real-time applications or meeting specific fill requirements. SA offers superior structural creativity for applications prioritizing puzzle interconnectedness over generation efficiency, though its unreliable performance under tight constraints limits practical deployment.

Related Work

cwc Crossword Compiler
Lars Christensen, 1999‑2002

An open‑source C++ constructor that fills a pre‑defined grid by **exhaustive letter‑by‑letter backtracking search** with dependency‑aware backtrack pruning and dictionary‑level optimisations; it focuses on speed and correctness of fill but leaves clue writing entirely to humans.

Quick Generation of Crosswords Using Concatenation
Dakowski, Jaworski & Wojna, 2024

Proposes an inductive approach that concatenates, rotates, and mutates smaller sub‑crosswords. Two variants: the first is **first‑/best‑improvement local search** and the second is **simulated annealing**, optimizing a cost function combining intersection count and letter density. Resulting puzzles meet structural constraints but are often too difficult for human solvers. The authors suggest future work on refined fitness metrics and cuckoo‑search.

A Fully Automatic Crossword Generator
Rigutini, Leonardo & Diligenti, Michelangelo & Maggini, Marco & Gori, Marco (2009)

Frames crossword generation as a **constraint satisfaction programming (CSP)** problem, where word placement is optimized under structural and lexical constraints. Their system also includes an **NLP-based clue extraction module** that automatically gathers definitions from web sources, enabling fully automated puzzle creation.

Generative AI Disclaimer

Portions of this paper were written with the assistance of generative AI.