# Final Report:

# Modeling Human Problem-Solving Heuristics in the Target Number Puzzle

Hayden Schennum
hschennum3@gatech.edu
CS 6795 - Georgia Institute of Technology
2025-11-15

***Abstract*—In this work, I investigate whether patterns in historical *Countdown* Numbers Round data can be used to identify heuristics that humans use to solve Target Number Puzzles; furthermore, I evaluate whether these heuristics can improve computational search algorithms. I built a web scraper tool to collect over 40 years of Numbers Rounds, separated into those solved by contestants within the 30-second limit and those that were not. I implemented several solvers: naive depth-first search variants (brute-force search methods) and heuristic-guided best-first search variants inspired by findings from cognitive science. Based on experiments, I found that heuristic-based solvers result in a larger performance gap between easy and hard problems than non-heuristic methods, suggesting that strategies similar to these heuristics are used by humans. The best solver, which prioritizes proximity to the target and uses factors of the targets as subgoals, achieved faster average solution times than standard memoized depth-first search. This study serves as an example of how cognitive insights can improve practical algorithm design. The project repository is available at: https://github.com/hschennum/CogSciFinalProject**

## I. Introduction

### A. Background

This cognitive science project involves topics from artificial intelligence and psychology; in particular, it concerns human heuristics used when solving the Target Number Puzzle (TNP). The TNP is a class of problems where a set of numbers must be combined via arithmetic operations to reach a target. The most popular variant of TNPs is the Numbers Round on the British game show *Countdown* [1] [2], e.g.

> Obtain the given target by performing operations on the numbers provided.
>
> You may use the four arithmetic operations: addition, subtraction, multiplication, and division. After each operation, the intermediate result must be a positive integer. Each number can only be used once. You do not necessarily have to use all the numbers.
>
> Numbers: $1, 1, 4, 5, 6, 7$ Target: $899$

in which one solution is $((6 * 5) + 1) * ((4 * 7) + 1) = 899$. This video shows example *Countdown* Numbers Rounds: https://www.youtube.com/watch?v=KvuDCVxLYz4&list=PLUpgiXuylHjfhwYE8SQFxd5lpmC09W5Ta (see the channel for additional Numbers Rounds). Other TNP-style problems (with somewhat different rules) are the 24-Game [3] and Make-N [4].

### B. Research Question

The question I address in this project is:

> To what degree can human heuristics in the Target Number Puzzle be identified from historical contestant responses, and to what degree can these heuristics be used to reduce the execution time to solve the puzzle via a computer's search algorithm?

Note that there are two parts involved: the question "whether the identified heuristics are actually being used by human contestants" and the question "whether the identified heuristics actually help a computer solve the TNP faster".

### C. Personal Interest

For several years, my family has ordered a daily puzzle calendar. Our favorite puzzle category is the TNP - we typically all solve it individually and then compare our solutions. One thing I noticed was that sometimes, a solution would "jump out" at one of us and they would solve it in a few seconds; the rest of us might take several minutes to solve the same puzzle. Furthermore, some puzzles seemed to take far less effort to solve than others. I decided it would be interesting to study what heuristics are used by humans to solve TNPs - understanding these heuristics would help explain why some puzzles are easier for humans than others.

### D. Importance

TNPs are worthy of study for the following reasons:

**TNPs require arithmetic fluency and creativity.** To solve a puzzle, a player needs to perform mental arithmetic, and they also need to formulate a novel solution approach - both are widely-important skills [5].

**TNPs highlight what humans pay most attention to.** Certain features of the available numbers are more salient than others, e.g. the knowledge that "899 is close to 900" is more salient than "899 is close to 898" - perhaps 900 is more familiar than 898 [6].

**TNPs are representative of a large class of problems.** The puzzle fits nicely into the existing framework of "Problem Solving: Search" in the field of artificial intelligence: there is an initial state, goal state, and available actions to transition between states [6] [7].

These features make it feasible to use a computational model (i.e. a search algorithm), compare the performance of the model to human contestants, and use this comparison to:

- infer the cognitive mechanisms being used to solve the puzzle, and
- use these mechanisms to improve the search algorithm.

### E. Cognitive Science Concepts

In general, a heuristic is a rule of thumb (mental shortcut) that contributes to finding a satisfactory solution without considering all possibilities [8] [9]. Humans and computers have fundamentally different ways of solving search problems: humans rely more on heuristics and computers rely more on algorithms [8]. The structure of computers makes them more suitable than humans for computational tasks, so a computer will solve a TNP much faster (roughly 10,000x faster) than a human [10]. In contrast, human problem-solving in a TNP more closely resembles the concept of bounded rationality, where limited computational power forces humans to use an approximate ("satisficing") approach [11].

The specific heuristics that humans likely use to solve TNPs are discussed in [5] and [6]; in this study, I use these heuristics to design unique "human-heuristic solvers" that are expected to perform better on problems quickly solved by humans. Notable heuristics include:

**Use large numbers early.** As mentioned in [5], solvers of the 24-game tend to use the largest and second largest numbers in the number set as part of the first operation. These are likely the most salient items in the number set upon first glance.

**Get close to the target early.** Both [5] and [6] note that TNP players tend to get as close as possible to the target in the first operation. Perhaps being "nearby" in numerical value is perceived as being "nearby" an actual solution.

**Use factors of the target as subgoals.** In both [5] and [6], an example is shown where a player realizes the target can be factored (e.g. $120 = 6 * 20$) and tries to build each of these factors using the available numbers. This can be interpreted as decomposing a problem into subproblems, a task that humans excel at.

## II. Model Design

### A. Scraping Countdown Games

To distinguish between "problems quickly solved by humans" and "problems not quickly solved by humans", I needed some way to measure the difficulty of TNPs. Existing sources only speculate which features of a TNP make it more or less difficult (see [1], [6], [10], [12]), but I wanted a concrete way to distinguish "easy" vs "hard" problems. To this end, I found a database of every *Countdown* game ever played since the show's inception in 1982 [13] - the database contains approximately 25,000 Numbers Rounds (TNPs). For example, here is the forum post corresponding to the Countdown video in the Introduction, for 11th Jan 2024 (Series 89): http://www.c4countdown.co.uk/viewtopic.php?t=16545.

I built a web-scraper tool to collect every TNP in the database. For example, here are the scraped records for the Numbers Rounds played on 11th Jan 2024:

2024-01-11;Round3;3,7,5,1,4,10;482;T
2024-01-11;Round6;50,100,5,3,7,4;879;F
2024-01-11;Round9;3,1,7,4,10,8;254;F
2024-01-11;Round14;75,100,10,2,8,9;520;F

The date and round number uniquely identify a Numbers Round, the numbers set and target define the TNP itself, and the final column is "T" iff **either contestant solved the problem exactly in the allotted 30 seconds**. I classify "T" records as "easy" problems and "F" records as "hard" problems. Easy problems and hard problems were split into their own separate datasets, so that each solver's performance could be split into a "performance on easy problems" measure and a "performance on hard problems" measure.

### B. Enumerating Possible Countdown Number Sets

In a Numbers Round, the 6 numbers are drawn from the following set; the contestant can choose how many large (>10) numbers to use, but cannot select the specific numbers:

$$\{1,1,2,2,3,3,4,4,5,5,6,6,7,7,8,8,9,9,10,10,25,50,75,100\}$$

This means there are 13,243 distinct number sets (see [1] [10] [12]). I enumerated these sets for use in validating the scraped data and in validating my algorithm implementations.

Note that the target number can be in the range 100-999 inclusive.[1] This means there are $13,243 * 900 = 11,918,700$ possible Numbers Round problems. Per [1], only 10,871,986 of these are solvable (for example, Numbers: $1,1,2,2,3,3$ Target: 700 is not solvable).

### C. Cleaning the Scraped Data

From the initially-scraped records, I kept only those records that matched the following conditions:

- **Problem must be solvable.** One of the contestants or the co-host must have provided a solution that matched the target exactly. In addition, their solution must actually be correct (there was at least one record in which the problem is actually unsolvable, but the co-host's answer was marked as correct).
- **Problem must be valid.** The problem must use one of the 13,243 possible number sets. Some rare records have illegal large numbers (e.g. 37) due to a special event, other records have three 6's or three 9's (likely due to a mistakenly-rotated card).

After cleaning, 21,074 scraped records remained. Of these, only 10,000 were actually used (for speed of obtaining results).

[1]There was some contention among sources over whether 100 could be the target number - [10] claimed that 100 is not a valid target, while [1] claimed that 100 is a valid target. Having scraped all the historical games, I can confirm that 100 appears as a target number in at least 7 Numbers Rounds.

### D. Tasks for Solvers

There are two tasks that each solver should accomplish:

**Exhaustive search of possible targets ("exhaustive-search task").** The input is the full list of 13,243 possible number sets, and the output is the time taken (s) to enumerate all reachable targets in the range 100-999 and the total number of solvable games. This task is done to validate my implementation of the solvers, to make sure that each algorithm correctly reports that exactly 10,871,986 of the 11,918,700 possible games are solvable.[2]

**First solution that reaches the given target ("first-solution task").** The input is the list of 10,000 scraped Countdown games (combinations of number set and target), and the output is the total time (s) taken to find a single valid solution for each game. Note that this is the task I primarily use to compare all solvers with each other, because this task (find just one solution) more closely resembles what a human does when solving a TNP.

### E. Implementing Standard Solvers

I implement several standard (non-heuristic) solvers as a baseline to compare with existing literature [10] and to compare with the human-heuristic solvers. The standard solvers are as follows:

**Depth-first search (DFS).** This is a classic search algorithm that explores a tree by exploring all the way down each branch before backtracking. Given that a node consists of a set of numbers, successor nodes can be generated by applying each of the arithmetic operations (+,-,*,/) to each pair of numbers [10].

**DFS with memoization.** Perform DFS, and also track which nodes (sets of numbers) have been visited so far, to avoid re-computing identical branches [10].

**Iterative-deepening DFS with memoization.** Perform DFS with memoization, but re-perform the entire search at a depth of 1, then 2, then 3, etc. - the idea is that a low-depth solution (i.e. one that uses fewer numbers) might be found faster [7]. Note that this algorithm was not included in [10].

### F. Implementing Human-Heuristic Solvers

The human-heuristic solvers, inspired by the corresponding heuristics mentioned in Section I-E, are as follows:

**Largest-first ordering.** Order the numbers from largest to smallest before applying them to the best standard solver. The idea is that combining larger numbers as the first operation might lead to a solution earlier in the tree exploration.

**Greedy best-first search (BFS) with proximity heuristic.** Instead of using DFS, use a priority queue that expands first the node that is "closest" to the target [7]. Here, the heuristic measure of "closeness" between node (number set) $S$ and target $T$ is the minimum distance from any number in the set to $T$:

[2]For efficiency, after the first algorithm was verified, I actually used a fixed subset of 1,000 number sets; exactly 817,443 of the 900,000 possible games must be reported "solvable" for an algorithm to pass.

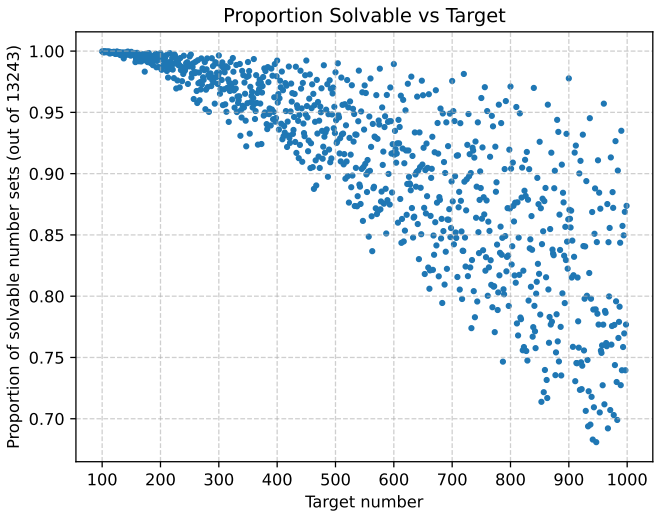


Fig. 1. Proportion of number sets (out of 13,243 possible number sets) that result in a solvable game for each target number. As target number increases, it is less likely that the problem is solvable.

$$h(S,T) = \min_{s_i \in S} |s_i - T| \tag{1}$$

**Greedy BFS with combined proximity and factor heuristic.** To account for using factors of $T$ as subgoals, the heuristic measure of "closeness" can be expanded to give a bonus when an element of $S$ is a factor of $T$. Adding a tunable $\alpha$ (factor bonus) which is larger when factors are more important, and a tunable $\beta$ (factor threshold) below which factors will not receive the bonus, the new heuristic for best-first search is:

$$h(S,T) = \min_{s_i \in S} \begin{cases} |s_i - T| - \alpha T, & \text{if } T \,\%\, s_i = 0 \text{ and } s_i \geq \beta, \\ |s_i - T|, & \text{otherwise.} \end{cases} \tag{2}$$

Note that all human-heuristic solvers also use memoization.

## III. Results

### A. Removing Largest-First Ordering

One early observation was that the "largest-first" ordering (or random ordering, or any other ordering) had negligible effect on solve time for all solvers in the first-solution task. This is likely because the optimal "first numbers to combine" for a given problem vary fairly-uniformly across problems (e.g. for some problems, it is best to start with $50 * 10$ to get close to the target; for other problems, it is best to start with $4 + 7$ to build an intermediate factor). Based on this, the largest-first ordering heuristic was dropped from further analysis; further discussion will only include the other five solvers.

### B. Results for Exhaustive-Search Task

As mentioned, the exhaustive-search task was primarily used to validate that my implementations were correct - each solver must first pass the exhaustive-search task with the

correct count of solvable problems before I adapt it for the first-solution task. As an initial validation, using the DFS with memoization solver, I re-constructed the plot of "proportion of solvable number sets vs target" (Fig 1); this plot exactly matches the equivalent plot in [1] and [10].

In addition, I recorded the amount of time each solver needed to solve a fixed subset of 1,000 number sets; the results are shown in Table I. It is no surprise that DFS with memoization has the smallest average solve time, for the following reasons:

- **Adding a "visited" cache (memoization) vastly reduces the search time relative to cache-less DFS.** The cache allows the "DFS with memoization" solver to skip number sets it has already seen before; these visited sets will only contain duplicate number sets that have been fully-explored elsewhere in the search tree.
- **The overhead from iterative deepening adds no benefit in the exhaustive-search task.** The point of iterative deepening is to quickly return a shallow solution that uses few numbers (given a target number); but when exploring all possible targets, the initial depth-limited searches cannot exit the algorithm early.
- **The overhead from BFS adds no benefit in the exhaustive-search task.** Similar to the above, the point of BFS is to exit the algorithm early based on the heuristic measure, but this requires extra overhead from a priority queue and heuristic calculation; when exploring all possible targets, BFS cannot exit the algorithm early.

It is notable that my average time per set is significantly higher than the expected average time per set from [10] - this is likely because my implementation was in Python (slower due to having interpreted code, dynamic typing, and more abstraction overhead), while the implementation in [10] was done in Ocaml (faster due to having compiled code, static typing, and less abstraction overhead).

### C. Results for First-Solution Task

Recall that the first-solution task is the main task I use to compare solvers with each other, because this task (find just one solution) more closely resembles what a human does when solving a TNP. Note that the "BFS with proximity and factor heuristic" solver has two parameters to tune: $\alpha$ (factor bonus) which is larger when factors are more important, and $\beta$ (factor threshold) below which factors will not receive the bonus. The idea behind $\beta$ is that small factors such as 1 should not make a set qualify for the bonus (since 1 divides all targets); only sets with tactical intermediate factors (such as factor 29 when target is 899) should receive the factor bonus.

The results of tuning $\alpha$ and $\beta$ are shown in Fig 2. For most values of $\alpha$, the optimal $\beta$ is 2 or 3 (so it is best to exclude 1 and/or 2 from allowing a set to qualify for the bonus); surprisingly, the globally-optimal values are $\alpha = 0.9$, $\beta = 1$. It appears that for $\alpha = 0.9$, it is beneficial to give the factor bonus to all number sets containing 1 - this would allow the algorithm to more quickly find the solution to something like: Numbers: $1, 2, 4, 5, 6, 100$ Target: 599 (despite 599 being prime, the solver would expand the solution $(100 * 6) - 1$ early).

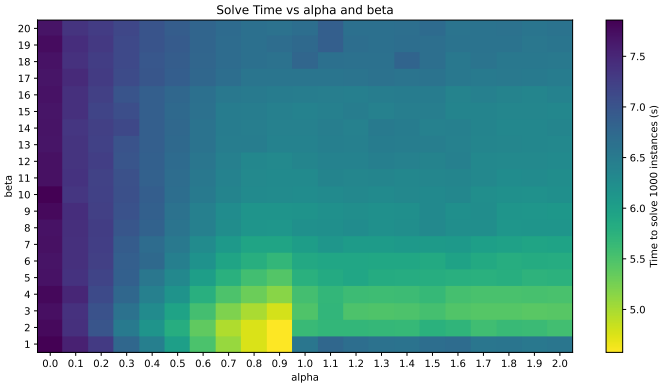


Fig. 2. Grid search for selecting best values of $\alpha$ and $\beta$. For each combination, time was measured on the first-solution task, using a fixed subset of 1000 problems comprising both T (easy) and F (hard) games.

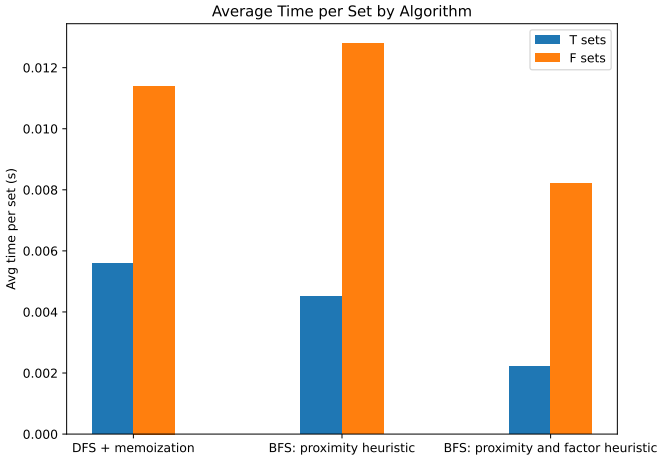


Fig. 3. Visualization of performance on first-solution task for easy (T) games and hard (F) games. BFS-prox-and-factor used $\alpha = 0.9$, $\beta = 1$.

With the optimal parameters found, I recorded the amount of time each solver needed to solve a fixed subset of 10,000 scraped *Countdown* games - split into performance on easy (T) games and performance on hard (F) games (Table II). For the best three solvers, the performance on easy vs hard games is shown in Fig 3. The results here are decisive:

- **All algorithms have better performance on easy problems than on hard problems.** Even heuristic-free solvers take less time on easy problems - likely because easy problems tend to have a solution with a shorter search depth (so on average, they will be found more quickly even without using human heuristics).
- **Adding memoization to DFS significantly improves performance.** As mentioned above, a cache of "visited" states allows the solver to prune nodes it has already seen, avoiding duplicate work.
- **Adding iterative deepening to DFS-with-memoization harms performance.** This is likely because the search

TABLE I
PERFORMANCE ON EXHAUSTIVE-SEARCH TASK. BFS-PROX USED TARGET=550; BFS-PROX-AND-FACTOR USED TARGET=550, $\alpha = 0.5$, $\beta = 11$ (NOTE THAT THESE PARAMETERS HAVE NO EFFECT ON THE EXHAUSTIVE-SEARCH TASK).

| Algorithm | Time to solve 1000 sets (s) | Avg time per set (s) | Expected avg time per set (s) from [10] |
|---|---|---|---|
| DFS | 453 | 0.453 | 0.012 |
| DFS + memoization | 29 | 0.029 | 0.005 |
| DFS + memoization + iterative deepening | 56 | 0.056 | not in [10] |
| BFS: proximity heuristic | 41 | 0.041 | not in [10] |
| BFS: proximity and factor heuristic | 43 | 0.043 | not in [10] |

TABLE II
PERFORMANCE ON FIRST-SOLUTION TASK. BFS-PROX-AND-FACTOR USED $\alpha = 0.9$, $\beta = 1$.

| Algorithm | Time to solve 10000 sets (s) | Avg time per set (s) | Time to solve 6484 T sets (s) | Avg time per T set (s) | Time to solve 3516 F sets (s) | Avg time per F set (s) |
|---|---|---|---|---|---|---|
| DFS | 370 | 0.0370 | 149 | 0.0230 | 221 | 0.0629 |
| DFS + memoization | 76 | 0.0076 | 36 | 0.0056 | 40 | 0.0114 |
| DFS + memoization + iter. deepening | 223 | 0.0223 | 99 | 0.0153 | 124 | 0.0353 |
| BFS: proximity heuristic | 74 | 0.0074 | 29 | 0.0045 | 45 | 0.0128 |
| BFS: proximity and factor heuristic | 43 | 0.0043 | 14 | 0.0022 | 29 | 0.0082 |

tree in a Numbers Round has a higher branching factor near the root of the tree (e.g. going from 6 to 5 numbers in a set involves ${}_6C_2 * 4 = 60$ branches) than toward the bottom of the tree (e.g. going from 2 to 1 number in a set involves ${}_2C_2 * 4 = 4$ branches). Hence the repeated depth-limited searches add far more overhead than the benefit of early stopping when a solution uses fewer numbers.

- **Relative to memoized DFS, BFS with proximity heuristic improves performance for easy problems, but worsens performance for hard problems.** This is strong evidence that humans use a proximity heuristic when solving a TNP - for easy problems, the proximity heuristic guides the solver to the correct solution earlier; while for hard problems, the proximity heuristic results in extra time being spent.
- **Augmenting BFS with both a proximity and factor heuristic further improves performance.** Performance on both easy and hard problems is best for this solver, so the factor heuristic was universally helpful for all problems. In addition, this is strong evidence that humans use a factor heuristic when solving a TNP - the ratio of average (F solve time)/(T solve time) is highest for this solver (3.7, as compared to 2.8 with just proximity heuristic), implying greater distinction between hard and easy problems.

## IV. DISCUSSION

The idea of applying a domain-specific heuristic to a search problem is nothing new [7]. However, this study takes this heuristic-application a step further by **comparing how the heuristic-based solvers perform on problems easily solved by humans versus problems that humans have difficulty solving**. Since this study shows that human-heuristic solvers (BFS with proximity heuristic and BFS with proximity+factor heuristics) have a wider performance gap between easy problems and hard problems, this study gives evidence that these heuristics (or something similar to them) are actually in use by the humans solving TNPs. In addition, for the first-solution task, the human-heuristic solvers (in particular, tuned BFS with proximity+factor heuristics) resulted in a faster average solve time than naive DFS with memoization, so these human heuristics did help a computer solve the problem faster.

This study is important for the following reasons:

**It shows how cognitive insights can improve practical algorithm design.** This study demonstrates a classic application of cognitive science: in understanding how the mind works, one can use this understanding to build more efficient algorithms. In this case, tuned BFS with proximity+factor heuristics achieved a $76/43 = 1.8$ speedup compared to naive DFS with memoization; this was only possible by hypothesizing which heuristics are used by humans.

**It helps explain why some puzzles feel easier than others.** In showing which heuristics are likely used by people when solving a TNP, this study allows a more concrete way to explain why certain puzzles are easier than others (beyond simply saying a solution "jumped out"). Puzzles in which the proximity and/or factor heuristics are most applicable will most likely correspond to easier problems for a human to solve.

**It validates algorithms using real-world behavior.** Rather than using small-scale tests and anecdotal evidence, this study uses scraped data from over 40 years of *Countdown* games. This allows a more believable (and more reproducible) ground truth in distinguishing "easy" versus "hard" games.

These insights (improving algorithms via cognitive understanding, increasing explainability of problem domain, using reality-grounded data) were applied to this specific study; but these same insights are also applicable to other studies. In particular, using an understanding of the mind to improve algorithms is a practical application of cognitive science.

## V. LIMITATIONS

With society's current understanding of the mind and current knowledge of algorithms, it is unfeasible to perfectly mimic

how a human solves a TNP [6]. This brings to light several limitations of this study:

**Humans don't use search trees to solve a TNP.** While it is true that humans must build the target number using operations from the number set, it is ridiculous to consider a human doing the following at each step:

1) Begin with a number set (node) of 6 numbers
2) Expand the node (apply all operators +,-,*,/ to each pair of 2 numbers), resulting in 60 possible next steps; add these in heuristic order of priority in a "frontier" data structure
3) Pick the highest-priority node from the frontier and expand it ... (continue until target found)

Humans likely use a more fluid, less rigorous approach that relies more on understanding the relations and patterns among numbers [6]. Hence, any attempt to model human behavior via a search algorithm will always be an approximation. A more accurate understanding of human processing would be needed to increase the accuracy of a mathematical model.

**The factor heuristic doesn't actually work backward from the target.** While this heuristic was inspired by humans' ability to decompose a problem into subproblems, it only works when the target can be directly factored. For example, consider the problem: Numbers: $3, 4, 8, 8, 9, 100$ Target: $797$. A human can quickly see that 797 is close to 800 (use the 3), then 8 and 100 are both available in the number set, leading to the solution $(8 * 100) - 3$. However, since 797 is prime, the factor heuristic will not apply to this problem. A better "human-like" search strategy might be able to work both forwards and backwards at the same time (note that in prototyping, I tried a bidirectional search [7], but the high branching factor near the root made this strategy ineffective).

Another limitation unrelated to the search algorithm is that:

**"Time to solve problem" would be a better measure than "whether problem was solved in 30s".** The *Countdown* database [13] does not list the time it takes for contestants to solve the problem - in fact, if you watch the show, the contestants don't buzz or indicate the exact time instant that they've finished solving. Adding a continuous time measure would offer a better measure of difficulty rather than just classifying "easy" vs "hard" problems, but such a continuous measurement doesn't seem to be available for historical *Countdown* games.

## VI. Conclusions and Next Steps

To address the original two-part research question, the first part was:

> To what degree can human heuristics in the Target Number Puzzle be identified from historical contestant responses?

Because the proximity-heuristic BFS solver and the proximity-factor-heuristic BFS solver resulted in a wider performance discrepancy between easy Numbers Rounds and hard Numbers Rounds (relative to naive DFS with memoization), this supports the claim that humans use heuristics similar to the proximity heuristic ("get close to the target early") and the factor heuristic ("use factors of the target as subgoals") when solving TNPs.

The second part of the research question was:

> To what degree can these heuristics be used to reduce the execution time to solve the puzzle via a computer's search algorithm?

On the first-solution task, when using tuned BFS with proximity+factor heuristics, this achieved a $76/43 = 1.8$ speedup compared to naive DFS with memoization. This represents a significant performance increase.

The following are ideas for improvements to this study that don't rely on waiting for more human-like algorithms or more granular data:

**Try alternative search strategies and/or heuristic measures.** One could experiment with variants of bidirectional search to imitate problem decomposition, without suffering from high initial branching factor. One could also try dynamic programming in place of DFS - split the target into disjoint subsets and meet in the middle. Furthermore, alternative heuristics could be identified and tested within my existing framework - this would require hypothesizing additional strategies humans use when solving TNPs.

**Refine algorithm implementation.** One could rewrite my program in C (instead of Python) to reduce overhead. In addition, in my memoization, I cache entire number sets in the "visited" data structure; [10] mentions that memory usage is reduced if a hash table is used, wherein an element in "visited" is actually the sum of the hash values of all numbers in the set. Finally, if one wanted to solve as many problems as possible in a given time period, one could divide the workload among many cores and run the solvers in parallel (as described in [10]).

## VII. Task List

The completed task breakdown is shown in Table III; the total workload is greater than 100 hours.

## References


[1] DataGenetics, "Countdown numbers game." Accessed: 2025-09-19.
[2] M. Wylie and D. Eadie, *Countdown: Spreading the Word*. Granada Media, 2001.
[3] 4nums, "Intro to 24 the game." Accessed: 2025-09-19.
[4] Puzzles and Riddles, "Target number game." Accessed: 2025-09-19.
[5] H. L. J. van der Maas and E. Nyamsuren, "Cognitive analysis of educational games: The number game," *Topics in Cognitive Science*, vol. 9, no. 2, pp. 395–412, 2017.
[6] D. Defays, "Numbo: A study in cognition and recognition," in *Proceedings of the Fluid Concepts and Creative Analogies Conference*, ACM, 1995.
[7] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. New York: Pearson, 4th ed., 2020.
[8] P. Thagard, *Mind: Introduction to Cognitive Science*. Cambridge, MA: MIT Press, 2nd ed., 2005.
[9] A. Goel and K. McGreggor, "Intro to cognitive science." https://gatech.instructure.com/courses/473194. Accessed: 2025-09-13.
[10] J.-M. Alliot and C. Vanaret, "(the final) countdown," in *Proceedings of the Global Conference on Artificial Intelligence (GCAI 2015)*, pp. 14–26, EPiC Series in Computer Science, 2015.
[11] H. A. Simon, *Models of Man: Social and Rational*. New York: Wiley, 1957.

TABLE III
TERM PROJECT TASK LIST

| Week | Task | Hours | Complete? (Y/N) |
|---|---|---|---|
| 3 | Read through assignment instructions and example projects | 2 | Y |
| 4 | Brainstorm term project ideas, select favorite idea | 5 | Y |
| 5 | Research known human heuristics used to solve the Target Number Puzzle | 8 | Y |
| 5 | Research existing implementations of the puzzle's solution | 8 | Y |
| 5 | Plan project approach | 10 | Y |
| 5 | Write project pitch document | 6 | Y |
| 5 | **PROJECT PITCH DUE** | | |
| 6 | Enumerate all 13243 possible Countdown number sets (for validation and benchmarking) | 3 | Y |
| 7 | Implement/debug scraper to collect human difficulty ratings of historic Countdown problems | 20 | Y |
| 7 | Clean/standardize scraped data | 5 | Y |
| 8 | Implement/debug standard solvers: depth-first, add caching, add iterative deepening | 15 | Y |
| 9 | Obtain results for standard solvers on "exhaustive search of possible targets" task; refine to ensure match with literature | 5 | Y |
| 9 | Obtain results for standard solvers on "first solution to reach target" task | 3 | Y |
| 9 | **OPTIONAL MIDPOINT CHECK-IN DUE** | | |
| 10 | Implement/debug human-heuristic solvers: best-first with proximity heuristic, add factor heuristic | 10 | Y |
| 11 | Obtain results for human-heuristic solvers on "first solution to reach target" task | 3 | Y |
| 12 | Compare performance of standard solvers and human-heuristic solvers; obtain plots | 5 | Y |
| 13 | Tune best solver (best-first with proximity+factor heuristics) to minimize mean solving time | 5 | Y |
| 14 | Write final report document | 16 | Y |
| 14 | **FINAL REPORT DUE** | | |
| 15 | Prepare final presentation poster | 8 | N |
| 15 | Rehearse presentation | 3 | N |
| 15 | **FINAL PRESENTATION DUE** | | |
| | **Total Hours** | **140** | |

[12] crosswordtools, "Numbers game solver faq," n.d. Accessed: 2025-09-19.
[13] Apterous, "Countdown database (cdb)," n.d. Accessed: 2025-09-19.