CS227: Reasoning Methods in AI
Assignment 2 Feedback for:

Harry Robertson Todd Sullivan Pavani Vantimitta

May 14, 2008

I Overview

Project 2 was designed to give everyone hands-on experience with solving classical constraint satisfaction problems. Every group implemented algorithms which allowed them to successfully fill in crossword puzzles, and some even solved the infamous Zebra problem. Overall, we were pleased to see improvements in discussion, engineering, organization, and presentation over the last report. Well done.

Section II provides general feedback and observations for all assignments. Section III provides specific feedback for your group. Additional feedback has also been left on a hard-copy of your group's report.

II General Feedback and Observations

1 Report Organization and Presentation

1.1 Good Overall Style

Most groups capitalized on the advice we offered in response to the previous assignment. We encourage you to continue to take advantage of that feedback and maximize how effectively you communicate your hard work and interesting results.

1.2 Code / README Line Length

Please do us a favor and keep the majority of lines to no more than 80 characters long. If you have the odd line which goes over 80, then that is fine, but in general long lines make a file painful to read. This is especially true in the README files – our terminal is only so wide!

2 Ideas to Ponder

2.1 Random Restarts

Groups which analyzed variance realized that the distribution of run times was heavily-tailed. In other words, a bad start could severely diminish the performance of an algorithm. Furthermore, these outliers occurred with some measurable frequency. To counteract this problem, two groups implemented random restarts as way of mitigating the damage done by a poor initial starting position. These groups achieved relatively high performance and their outliers were less extreme. As described in Section 3, the two best performing and most consistent teams utilized this strategy.

2.2 Puzzle Solution Differentiation

Groups presented a variety of ways for obtaining different solutions to a given puzzle. This ranged from backtracking from a good solution and using a tabu list to restarting from the beginning with a new random seed. We observed that most groups used the dictionary in lexicographic order. One simple approach to achieve very different and more interesting solutions could have involved sorting the dictionary in some new, arbitrary order.

One group added an option to have the solver never reuse any word in a given solution to a puzzle. They went even further and measured its impact on performance (almost negligible as it turns out). This was a neat idea and definitely resulted in more interesting solutions to the puzzles.

2.3 Dictionary Perturbations

It would have been interesting to see how the algorithms were affected by perturbations to the dictionary. For example, in what circumstances would a smaller or larger dictionary be more helpful?

3 Performance

This section gives an overview of how each team performed when using their best-performing algorithm. There were a wide range of results with many orders of magnitude separating the most and least efficient solutions. The best solutions solved all crosswords puzzles in consistently less than a second and never exceeded the timeout threshold.

3.1 Method

- **Overview.** We collected our own data using your "best-performing" solvers on each of the four provided puzzles. 100 trials were run with each team's solver on each puzzle. The projects were compiled as directed by each group's `README`. Experiments were run on the `pod` cluster.

- **Algorithm Selection.** The "best-performing" algorithm was the combination of features (forward checking, etc.) which resulted in the fastest *reported* median run times and the fewest timeouts. Furthermore, the algorithm was chosen on a per-problem basis. In other words, if you had one combination run best on puzzle 1 and a different one run best on puzzle 2, then the appropriate combination was used for each puzzle.

- **Fairness**. Teams and puzzles were run in a round-robin fashion to maximize fairness. In particular, it improved chances that over 100 trials every team would see a similarly loaded machine (throughout the collection of the results, an otherwise idle machine was used). Furthermore, rotating puzzles and teams made sure the cache was cold at the beginning of any trial.

- **One Trial at a Time**. When necessary, the submitted code was modified to only run a single trial on a single puzzle at a time. This ensured that no team's run time would be overestimated due to the solving of multiple puzzles instead of the one intended puzzle per trial.

- **Randomness**. We appreciate that some groups took our advice and added a mechanism for achieving a deterministic result. However, for the purposes of us measuring performance from scratch, we needed to ensure any randomness your algorithm utilized was not set from a specific, pre-determined seed. When necessary, the submitted code was modified to use a random time-based seed instead of a pre-determined, hard-coded value.

- **Measurements**. Run times were measured with the UNIX `time` utility. The run time for a given trial was computed as the sum of the CPU time spent in both user and kernel space. This further minimized the impact of any other activity on the machine.

- **Timeouts**. A timeout of ten minutes was enforced. Any program which had a different timeout was modified to use a ten minute timeout. The median and maximum run time values presented below do *not* include trials which timed out. However, all other values do include timeouts as a flat ten minute run time. This *underestimates* actual run time of programs which timed out, but provides a reasonable basis for comparison here. Finally, any trial which ended without finding a solution or tried to run for longer than the timeout was also assigned the standard ten minute timeout as their run time.

3.2 Results

These results have been anonymized by using team numbers instead of team member names. The team numbers were chosen in order of performance, e.g. the best performing team is referred to as *Team 1*. Though these often correlate with grades, this is not necessarily the case as grades are determined primarily by your *analysis*, not performance. These results exclude one team whose solver could not complete the required puzzles.

Specific feedback regarding your team's performance is located in Section III. That section also identifies which team number represented your team here.

Figures 1 and 2 compare the median and mean performance achieved by each team on each of the four puzzles with their best-performing algorithm. Table 1 contains the details about each teams' performance on all of the metrics we evaluated. In particular, it presents median, mean, best, and worse performance, sample standard deviation between trials (s), and the number of timeouts which occurred (or program crashes).

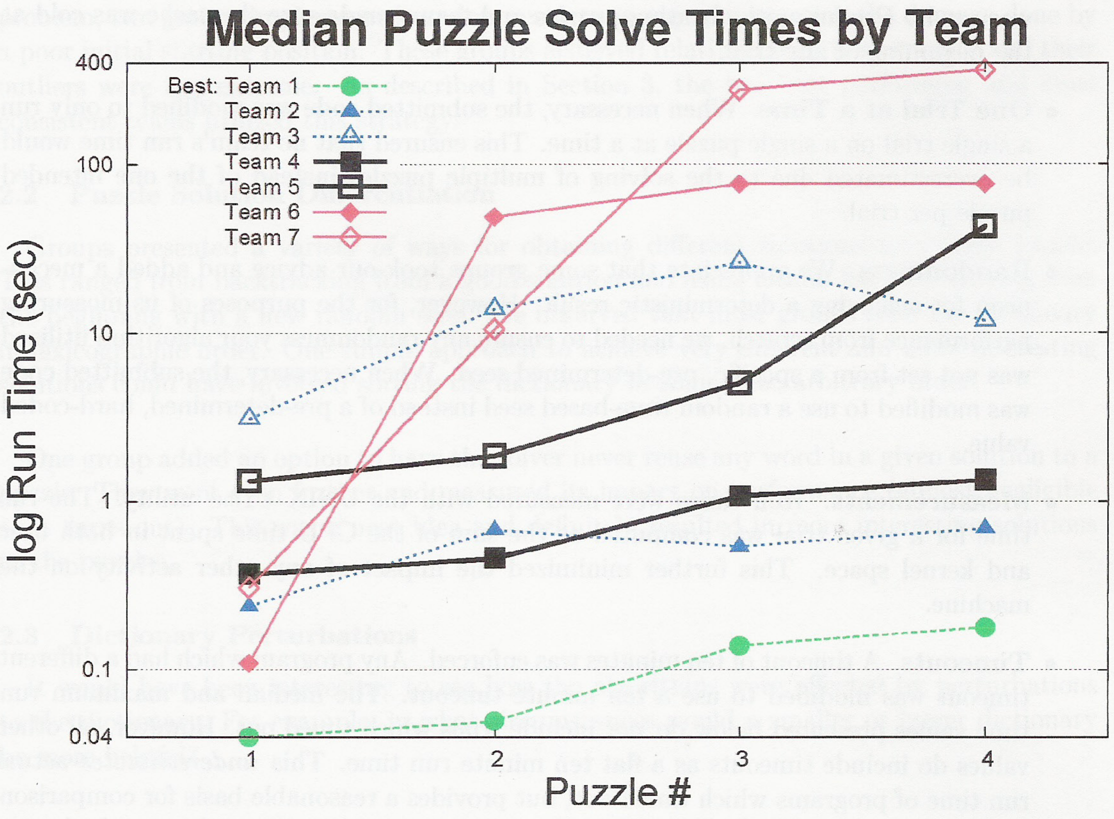


Figure 1: The log-scale run time axis shows the considerable difference between the different algorithms.

Interestingly, performance was distributed over a very wide range. The best performing teams typically solved even the hardest puzzles in under a second. They benefited greatly from the random restarts mentioned in Section 2.1. The difference between the most and least efficient solutions was approximately three orders of magnitude.

The sample standard deviation (s) correlated very nicely with performance. The teams which had the least variance also performed the best. Figure 3 provides a visualization of the amount of time it took to run each trial per team per puzzle.

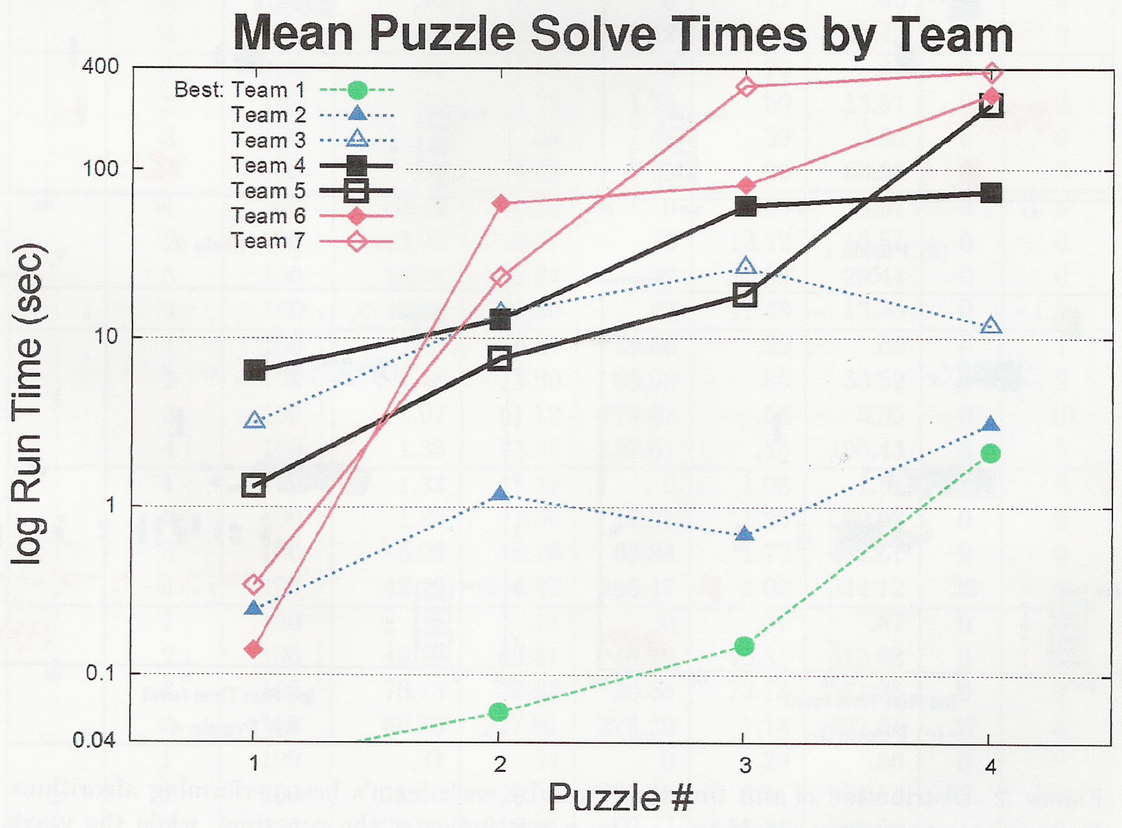
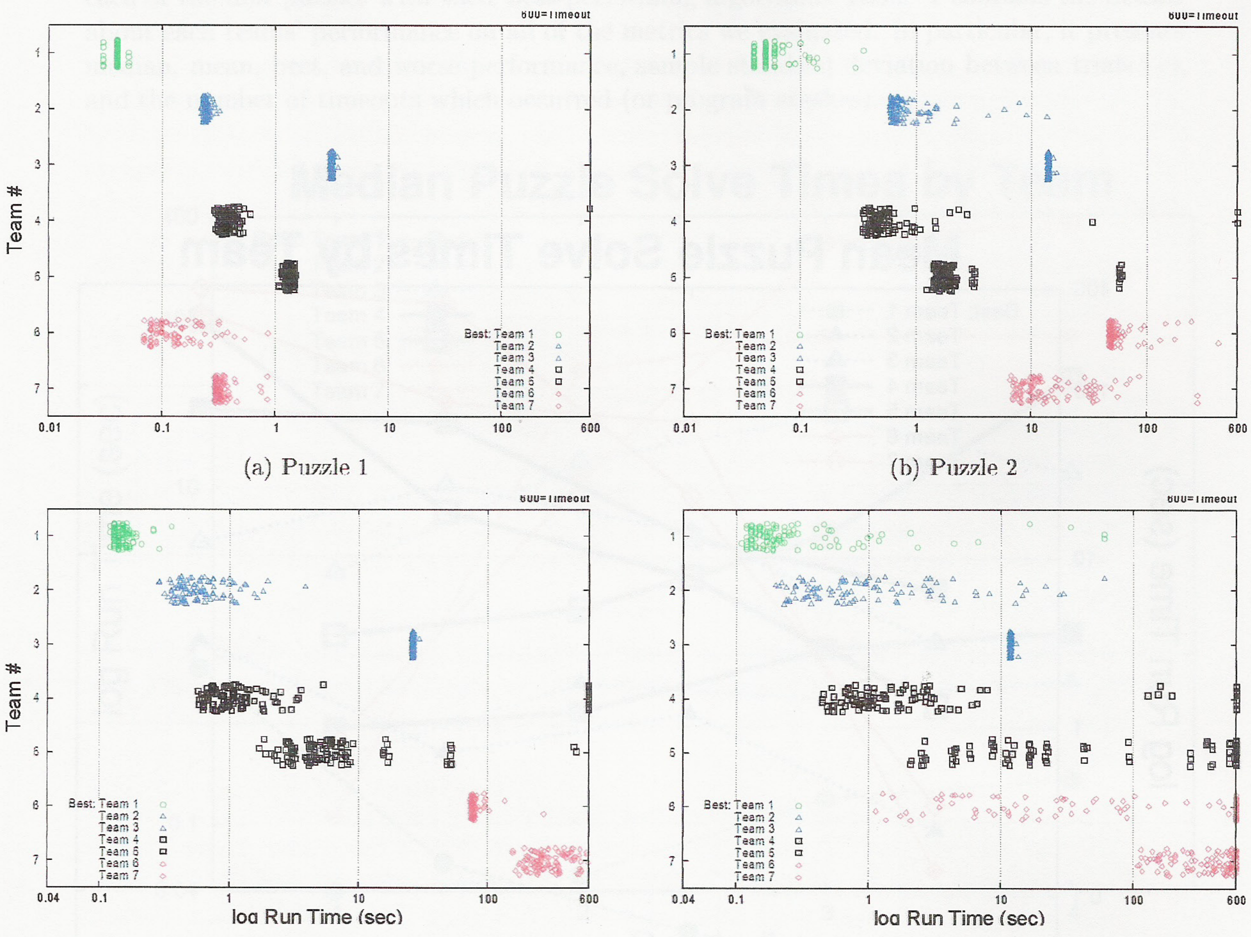


Figure 2: By plotting mean run times, we see that the most consistent algorithms benefit the most since the distribution of run times was generally heavily-tailed towards longer run times with little or no tail towards shorter run times.



(a) Puzzle 1 (b) Puzzle 2 (c) Puzzle 3 (d) Puzzle 4

Figure 3: Distribution of run times achieved by each team's best-performing algorithms. Each point represents a single trial. The x-axis indicates the run time, while the y-axis indicates the team which the result belongs to. The y-axis value is staggered around $y = 1$ for team 1, and so on. The staggered y-axis value is simply to improve readability for a given team's trial. We see that results are tightly clustered for the easier puzzles - there was not much difference from one run to the next. The deviation between runs on the fourth puzzle is particularly striking. The very long tails of the distribution show up occasionally even for the best implementations. Finally, team 3's results seem to be particularly well-clustered. This may be a fluke with how much randomness they inserted into their algorithm, but there certainly was some randomness as they solved the puzzle in a number of different ways (though they had many more repetitions than other groups, as most teams had no or few repetitious solutions).

Performance by Team

Team	*Puzzle*	#	*Run Time (sec)*					#	#
		Trials	*Median*	*Mean*	*s*	*Min*	*Max*	*TO*	*Crash*
1	1	100	.04	.03	0	.03	.05	0	0
	2	100	.05	.06	0	.04	.35	0	0
	3	100	.14	.15	0	.12	.36	0	0
	4	100	.19	2.11	9.12	.11	60.42	0	0
2	1	100	.24	.24	0	.22	.32	0	0
	2	100	.68	1.17	1.75	.59	13.51	0	0
	3	100	.54	.68	.44	.29	3.90	0	0
	4	100	.69	3.08	7.73	.20	60.56	0	0
3	1	100	3.12	3.13	0	3.04	3.57	0	0
	2	100	13.97	14.01	.33	13.72	16.51	0	0
	3	100	26.18	26.24	.37	25.69	29.44	0	0
	4	100	11.78	11.80	.22	11.48	13.45	0	0
4	1	100	.37	6.37	59.66	.29	.60	0	1
	2	100	.46	12.90	83.93	.35	33.52	0	2
	3	100	1.07	61.12	179.62	.58	5.35	0	10
	4	100	1.33	73.87	187.61	.45	196.43	3	8
5	1	100	1.32	1.32	0	1.08	1.51	0	0
	2	100	1.86	7.56	17.01	1.25	60.68	0	0
	3	100	5.03	18.19	65.84	1.73	481.61	0	0
	4	100	42.29	244.92	266.47	2.09	544.12	28	0
6	1	100	.11	.14	0	.07	.87	0	0
	2	100	49.19	62.61	46.19	46.81	413.08	0	0
	3	100	76.73	80.83	20.65	73.73	271.35	0	0
	4	100	86.83	277.01	278.30	1.14	592.61	37	0
7	1	100	.32	.34	0	.29	.86	0	0
	2	100	11.40	22.98	38.83	6.51	274.11	0	0
	3	100	272.05	312.85	110.41	156.34	586.83	1	0
	4	100	397.40	378.80	171.31	108.99	593.70	15	0
AVG	1	700	.32	1.65	8.52	.03	3.57	0	1
	2	700	1.98	17.33	26.86	.04	413.08	0	2
	3	700	6.06	71.43	53.90	.12	586.83	1	10
	4	700	11.68	141.65	131.53	.11	593.70	83	8

Table 1: Complete results for each team with their best-performing algorithms on each puzzle.

III Feedback for the Robertson-Sullivan-Vantimitta Group

1 The Good

- **Good Analysis**. Your team provided an excellent analysis of the results. We were interested by your analysis of domain representation trade-offs as well.
- **Standard Deviation Measurements**. We were glad to see that you carefully looked at standard deviations in addition to the averages. Perhaps knowledge of the high variability led you to implement random restarts.
- **Random Restarts**. Random restarts was an great idea given the variance of the run times most teams encountered. The inclusion of this feature no doubt made your implementation one of the best.
- **Excellent Engineering and Performance**. Your performance on all of the puzzles was outstanding. Your careful analysis of the memory component was a welcome addition to the report, especially since you provided the empirical data needed to back up your claims. Your optimization of the arc consistency feature was impressive, though as you noted it is only a significant factor in the smaller puzzles that we tested. Of course, your run times are fast enough that maybe it is on the critical path even for the tougher puzzles you solved.

2 Areas for Improvement

- **Analysis of Solver w/o FC**. Forward checking was expected to provide a performance boost, so it is surprising that your performance on puzzles 3 and 4 actually declines when it is enabled. It would have been interesting to have seen an explanation for this phenomenon.
- **Results Presentation**. It would have been interesting to have seen the median result plotted as well. Furthermore, it was a bit difficult to pick out the the best solver for each puzzle from your graph. It would have been helpful to have denoted the winners and labeled them with their run times. It may have also been helpful to have labeled the others, though perhaps this would have created too much clutter.

3 Your Performance

In Section 3, you were Team 1. The data we collected for with your team's program is replicated below. Your implementation really made an impression on us and blew away your competition. The fact that you paired this excellent work with a good analysis ensured your success on this project. Great job!

4 Late Days

Please see Table 2 for information about the number of late days you have used and remaining.

Team 1's (Your Team's) Performance								
Puzzle	*# of Trials*	*Run Time (sec)*					*# of Timeouts*	*# of Crashes*
		Median	*Mean*	*s*	*Min*	*Max*		
1	100	.04	.03	0	.03	.05	0	0
2	100	.05	.06	0	.04	.35	0	0
3	100	.14	.15	0	.12	.36	0	0
4	100	.19	2.11	9.12	.11	60.42	0	0

Assignment	**Number of Late Days**
Boolean Satisfiability	0
Constraint Satisfaction	2
Remaining	5

Table 2: Late Day Statistics.

5 Grade

Your letter grade on this assignment: *A+*