A Study of Search Methods: The Effect of Constraint Satisfaction and Adventurousness

Hans Berliner
Gordon Goetsch

Computer Science **Department**
Carnegie Mellon University
Pittsburgh, PA. 15213

Abstract

This research addresses how constraint satisfaction interacts with the search mode, and how the ratio of breadth of effort to depth of effort can be controlled. Four search paradigms, each the best of its kind for non adversary problems, are investigated. One is depth first, and the others best first. All methods except one highly informed best first search use the same knowledge, and each of these methods is tested with and without the use of a constraint satisfaction procedure on sets of progressively more difficult problems.

As expected, the most informed search does better than the less informed as the problems get more difficult. Constraint satisfaction is found to have a pronouncedly greater effect when coupled with the most informed algorithm. Large performance increments over A* can be produced by the use of a coefficient associated with the *h* term, and this algorithm produces solutions that are only 5% worse than optimal. This is a known phenomenon; however, the range of this coefficient is very narrow. We term this coefficient, which controls the ratio of depth of effort to breadth of effort, the *adventurousness coefficient.* The less tractable a problem the greater the adventurousness should be. We present evidence to support this.

Introduction

Heuristics are employed when the domain space being explored is too large to search exhaustively. A heuristic increases **the** likelihood of making a correct choice, but cannot prevent **the** making of an incorrect choice. The knowledge embodied by **the** heuristics is needed to reduce the cost of the search, but is insufficient to alleviate the need to search. The basic problems associated with heuristic search are: the desire to follow "successful" branches, while leaving less successful ones **for** later, and, when to quit pursuing a branch as its estimated merit declines.

Both problems are addressed by any search paradigm; however, the second problem can be effectively dealt with by a constraint satisfaction procedure that eliminates states that can no longer be solved. Brute force methods solve problems by searching to **the** maximum penetration allowed in the time available. As **the** problems get more difficult, the utility of such methods decreases. Thus depth-first searches are often augmented with techniques such as branch and-bound and minimal move ordering knowledge, but the use of heuristic knowledge is minimal. In contrast, a pure best first search relies exclusively on its heuristic knowledge with all the search control decisions being based on that knowledge.

We wished to study the interaction between heuristics, search techniques, and constraint satisfaction. Superpuzz, a solitaire puzzle that can benefit from constraint satisfaction techniques, was chosen as our problem domain. We selected four search algorithms as being the best exemplars of their class for solving non-adversary problems, and devised a constraint satisfaction procedure. We then investigated the degree of degradation fcr each search algorithm as the problems became more difficult, and the interaction of constraint satisfaction with each technique.

The Domain

The domain chosen for this study was Superpuzz, an extremely difficult solitaire puzzle. The rules of Superpuzz are as follows:

Superpuzz is played with 24 cards, 6 (numbered 0 to 5) in each of 4 suits. To start a problem deal the cards in a raster of 6 wide by 4 deep. Then remove the "0" denomination cards, leaving "holes". Legal moves consist of moving a card into a hole, thus creating a hole at its former location. The card that is moved into a hole must be of the same suit as the card to the left of the hole, and be one higher in denomination.

No card can be moved to the right of the last card in a suit, nor to the right of a hole. If a hole is on the left edge, any 1 may be moved there. The game is won when all the cards have been placed in ascending order by suit, with one suit in the first 5 places of each row as demonstrated by Figure 1B. There is no requirement to have particular suits in particular **rows. The game** is lost when there are no longer any legal moves. These are **the** rules for 4x6 Superpuzz; it is also possible to play harder versions adjusting the rules for 4x7 and **4x8** formats.

	0	1	2	3	4	6
W	S3	H2	D2	D4	--	D6
X	--	C4	H3	C6	C1	--
Y	D3	S1	D1	S4	S6	--
Z	C2	C3	H4	H1	S2	H6

(A)

0	1	2	3	4	6	
D1	D2	D3	D4	D6	--	W
H1	H2	H3	H4	H6	--	X
C1	C2	C3	C4	C5	--	Y
S1	S2	S3	S4	S6	--	Z

(B)

Figure 1: An initial and terminal configuration

Figures 1A and 1B show an initial and terminal position respectively. A solution to the initial position can be found at the end of this article. The most challenging aspect of Superpuzz is determining which ace to move into a hole on the left-edge. Making the proper ace move is often-non intuitive, which makes the domain interesting and results in programs that outperform well practiced humans.

The Search Paradigms

Initial studies determined the following search algorithms were the bet,t exemplars of their class :

1. A depth-first search (Df) using the branch and bound, and iterative deepening |5| techniques. Iterative deepening has recently beer) proved to dominate simple depth first search when the depth of the solution is unknown |3| The constant of iteration used was 2. The bound is the number of misplaced cards (N) in the present configuration If the present configuration is at depth D, it is impossible to reach a solution at a depth less than D + N.
2. The A* search [4) A* expands the frontier node with the minimal function *l,* where / = g + h = D + N.
3. A best first search (BF1) that uses the simple evaluation function *f* - N + D. where , the adventurousness coefficient which we discuss later, was equal to 1.8.
4. A best first search (BF2) with a highly informed evaluation function that would encourage the development of good "positional" formations that could be transformed into wins.

A hash table containing the generated nodes plays a key role in three ways. In the best first searches, it becomes the representation of the tree. The hash encoding detects identical states, so that the same subtree will be searched only once. The hash encoding also detects cycles. In the depth-first search, the hash table only performs the two latter functions.

Two evaluation functions were required. These are 1) A misplaced card counter, and 2) A position goodness function. These functions are described in detail in [2]. The misplaced card function is used in all the search programs, while the goodness function is used only in BF2.

The Constraint Satisfaction Method

Any state of a domain is either solvable or unsolvable. We define the set of totally solvable domains to be those in which all states accessible from a solvable state are solved or solvable. In such domains any operator applied to any state preserves the solvability of the new state. Frequently, for each operator there exists a reverse operator that can re-establish the previous state. This type of problem is represented by puzzles such as the 15 Puzzle and Rubik's Cube. An alternate condition is that the permissible operations do not allow transformation to an unsolvable state.

Set against the class of totally solvable problems is the class of partially solvable problems in which not every state of the domain can be solved, and the set of operations allow unsolvable states to be reached from solvable states, For totally solvable domains the only thing of interest is the speed of the solution process and the quality of the solution. Tor partially solvable problems, each instance may have to be classified as solvable or unsolvable.

Constraint satisfaction is the term used for the sot of algorithms that can determine when a subtree cannot contain a solution. We wanted to study the role of constraint satisfaction on a difficult, partially solvable problem as the difficulty of the problem varied. This was the reason Superpuzz was selected, as standard puzzles such as the 15 Puzzle are totally solvable, and others such as Instant Insanity are not very difficult for a computer.

Both totally and partially solvable problems can use heuristic knowledge in order to speed up the search for a solution. Heuristic knowledge can be used to choose the order of applying operators and to evaluate the new states. However, totally solvable problems need no process to identify subtrees in which no solution can exist because, *ipso facto,* such subtrees cannot exist.

The constraint satisfaction function developed for this problem [2] is as follows: Once all the aces are in place, the final destination of every card can be determined. Cards not at their final destination must be moved. We only examine cards that are to the left of their destination since they are typically the hardest to move, If such a card is unmovable, the problem instance is unsolvable. The constraint satisfaction procedure is able to reject about 50% of all configurations presented to it during a search as being unsolvable Constraint satisfaction is applied only after all the aces are in place, since identifying deadlocked positions earlier was unproductive. It ts able to deal with situations where not only individual cards, but whole trains of cards must be moved. Trains arise frequently as the result of putting a card behind its predecessor. The problem of determining whether a tram can be moved is very difficult, and a number of finesses were used which are described in the above cited reference.

Results

Each of the four search algorithms was tested with and without deadlock detection on the same 100 randomly generated instances of 4x6, 4x7 and 4x8 puzzles. The programs were to determine if each problem was solvable (and give a satisficing solution) or unsolvable. Testing method details are given in [2].

Solvable	Deadlock				Without Deadlock			
	DF	A*	BF1	BF2	DF	A*	BF1	BF2
Size = 6								
Av. Sol. Ln.	22	22	23	26	22	22	23	26
Av. Nodes	4637	5425	1907	1563	4845	5711	2896	2912
Av. Time	18.9	18.7	8.6	6.3	19.2	18.5	11.3	12.0
No. Intract.	0	1	0	0	0	1	1	1
Size = 7								
Av. Sol. Ln.	33	33	31	36	32	34	32	37
Av. Nodes	22865	23962	8401	5469	23759	25923	10140	8376
Av. Time	107.2	96.3	34.6	23.0	82.8	91.8	36.9	33.7
No. Intract.	5	7	2	1	5	8	3	3
Size = 8								
Av. Sol. Ln.	56	61	47	47	55	65	48	60
Av. Nodes	53507	68254	22361	9961	55318	62705	24603	15146
Av. Time	269.2	258.2	98.0	43.3	226.6	254.8	97.4	60.6
No. Intract.	18	26	9	2	19	29	10	6

Table 1: Performance Statistics

Table 1 shows the average solution length, the average number of nodes per solution, the average time to complete, and the number intractable for the eight programs for each of the three problem widths. As expected, effort in all categories varies with problem difficulty. It should be noted that the A*, BF1, and DF algorithms all use the same knowledge, A* and DF for bounding purposes and BF1 as a simple measure of goodness, yet the BF1 algorithm requires fewer resources (nodes and time). While BF1 produces solutions comparable to A* and DF on smaller problems, it clearly outperforms the other two algorithms on the 4x8 problem set. The most knowledgeable search (BF2) clearly outperforms the others in all criteria, except solution length. Although BF2 is only slightly superior among the 4x6 algorithms, it has achieved an overwhelming dominance by the time the 4x8 puzzles are considered.

	Nodes			Times		
Search	Size Change			Size Change		
Technique	6:7	7:8	6:8	6:7	7:8	6:8
Deadlock						
DF	4.93	2.34	11.54	5.67	2.51	14.24
A*	4.42	2.43	10.74	5.15	2.68	13.81
BF1	4.42	2.66	11.76	4.02	2.83	11.40
BF2	3.50	1.82	6.37	3.65	1.88	6.87
No Deadlock						
DF	4.90	2.33	11.42	4.31	2.74	11.80
A*	4.54	2.47	10.98	4.96	2.78	13.77
BF1	3.50	2.43	8.50	3.27	2.64	8.62
BF2	2.88	1.81	5.20	2.81	1.80	5.05

Table 2: Performance Ratios

Table 2 shows the ratio of increase in resources for each algorithm type as the problem width increases. The data show that, in general, the more informed the algorithm is, the less the degradation in performance as the problem gets more difficult. This is an expected result: the trees grown by the most informed algorithm grow at a lower exponential rate than those grown more brute-force-like algorithms. The BF2 search has the most information. The next, BF1 has had its coefficient tuned so it is more responsive to its sole performance measure (misplaced cards) than the remaining algorithms, A* and DF which have the least information.

	Nodes			Times			No. Intract.		
	Size			Size			Size		
Search	6	7	8	6	7	8	6	7	8
DF	.96	.96	.97	0.98	1.29	1.19	---	1	.95
A*	.95	.92	.93	1.01	1.05	1.01	1	.88	.86
BF1	.66	.83	.91	0.76	0.94	1.01	0	.67	.90
BF2	.54	.65	.66	0.52	0.68	0.71	0	.33	.40

Table 3: Ratio of Effort with and without Deadlock Detection

Table 3 shows the performance ratio of a search method with deadlock detection, to the same method without deadlock detection. Ratios less than 1.0 indicate improvements, otherwise the additional work was not beneficial. As expected the deadlock detection improves performance on the nodes measure in all categories. The deadlock detection did not always result in a time savings. Deadlock detection reduces the number of intractable problems, especially for the BF2 search. BF2 makes best use of the constraint satisfaction process. This is an important result which we discuss in the next section.

Discussion and Conclusions

Superpuzz is a much more difficult than standard puzzles. Further, the difficulty of the game varies with the width of the puzzle. In this study we examine the 4x6, 4x7 and 4x8 games. Although the branching factor in each of these remains the same, the solution depth and percent of unsolvable problems increases significantly with increases of width.

The solution process can be thought of as occurring in two phases. In phase one, the combinatonc power of search attempts to sec whether it is possible to obtain a position where till four aces are in place. When this has occurred, the deadlock detection algorithm is invoked, which can reject about 50% of all positions it encounters. Phase two is invoked for positions that pass the deadlock test. Here by relatively small searches, the solution is either found or rejected. When no solution is found in the sub-tree, phase one again obtains control.

Given that the combination of deadlock detection and very small searches in phase two is very efficient, certain ideas emerge. If the position is solvable, then it is advantageous to reach phase two as quickly as possible. The BF2 search does this most effectively. It is not at all unusual to have the first all aces in-place configuration discovered by the BF2 search be solvable, whereupon the solution proceeds immediately. In those cases where this does not happen, the first dozen or so attempts do usually yield a solvable phase two. Only in cases where the solution is very contrived, or where there is no solution, is the BF2 procedure outperformed by others, in the case where there is no solution, all phase two positions must be explored and the procedure that reaches these with the minimum amount of effort is the most effective. From this it can be seen that some knowledge of what percentage of problems is solvable is instrumental in deciding on a search paradigm. In this research, the BF2 evaluation function is expensive to compute, as is traversing the tree. But, while the BF2 search is only marginally superior in the width 6 puzzle, it becomes completely dominant by the time the width is increased to 8.

Let us consider why one search paradigm is better than another. A* and DF are really quite similar. They probe to new depths in a breadth-first style that takes advantage of certain efficiencies. A* knows about effort remaining and builds a permanent copy of the tree, which it continues to expand at the best leaf nodes. DF also knows about effort remaining and gets its power from great efficiency in space and time. However, neither is able to venture very far on a probe down a branch unless it is continuously having success (reducing the number of misplaced cards). Any failure to improve this measure would immediately force the A* search to try another branch. Because the DF search has an iteration constant of 2, two non successful moves can occur in expanding a branch before returning. Since the criterion for success is rather simplistic, and it is very likely that any solution will require a number of non-constructive or backward-appearing steps, it is unlikely that either search will be able to make significant forward progress through such territory. Instead they plow steadily forward until the treacherous territory is overcome by *all* highly evaluated branches, and then pursue one to a successful conclusion.

Now consider the BF 1 search paradigm The evaluation function for BF1 is / - /i N + D. The constant /i is the advonturousness coefficient; for this program μ = 1.8. In BF1 the value of a descendant node either decreases by .8 units in case the number of misplaced cards is reduced, or is increased by 1.0. This allows the descendant node to put some distance between it and its competitors when it is able to take a few constructive steps intermixed with some that do not appear so constructive. The essential point is that a branch does not have to produce "progress" on every move The degree of such adventurousness is what the constant 1.8 controls, and for the given domain and evaluation function it appears to be best.

Best first search disciplines exist that have a reluctance to abandon a branch until it is judged a constant amount worse than the current best branch. The adventurousness coefficient allows the *number of non-intuitive moves included in* a *branch to be a linear function of the number of "good" moves.* This appears to be a better construction.

The BF2 search is even more adventurous (though it not clear how to obtain its adventurousness other than by empirical observation) since it can gain numerous points in heuristic value by placing a card into what is considered an advantageous location. This allows it to penetrate deeply in certain branches that it "likes" while leaving others behind. This additional knowledge appears to pay off in performance.

In some cases the evaluation function will lead the search up a blind alley. Here is where constraint satisfaction helps the most: *it can disenchant the search causing it to look elsewhere.* This happens for all of the searches, but is most effective in BF2 because the other searches are not as adventurous.

In any search paradigm, once a node is known to be deadlocked, its successors will never be expanded. However, these savings can only be realized once such a sub tree is reached. This is where adventurousness is important. If situations where constraint satisfaction procedures can be applied occur only after almost all important branches have been pushed to the same depth, then the savings will not be very great. Here, the most adventurous search has a big advantage (see Table 3) since // *allows selectively approaching the point where constraint satisfaction can be applied.*

If the above notions of adventurousness are correct, then the less tractable the domain, the higher the adventurousness should be We tested this hypothesis by re-running the? Bf 1 (deadlock) algorithm on all the 4x8 problems with μ = 2.4 This resulted in the average nodes per problem being reduced by 15%, and in the number of intractable problems being reduced from 9 to 5. We intend to investigate this scaling of adventurousness further in future studies.

In anv domain the heuristic function must evaluate the desirability of the moves available. The decision of whether to abandon a branch in a best first search or continue it is basic to the efficiency of the search. The adventurousness coefficient for any domain /function combination, determines the degree to which the past history of the branch influences this decision. It is not necessary to "succeed" on every move in a branch in order to continue it. Instead a success gradient (the adventurousness coefficient) must be maintained. This tends to produce consistent, plan-like behavior.

A minimal solution to Figure 1A is: (read left to right; names of moving cards only, except for aces where the row is also given).

C2 S1 (Z) D4 D3 D4 CI (Y) C2 D1 (X) C3 S2 H2 S4 C4 D2 S5 C5 H4 S3 D1 (W) H1 (X) S4 D2 H2 S5 D3 D4 D5 H5

Acknowledgements

We wish to acknowledge the efforts of T. Anantharaman, B. Pearlmutter, and H. Printz who were the first to use constraint satisfaction in Superpuzz, and K. Goldberg whose graphic support aided in understanding the performance of the search.

References

[1] Berliner, H., "On the Construction of Evaluation Functions for Large Domains", Proc. IJCAI-77, Tokyo, 1977.

[2] Berliner, H., and Goetsch, G., "A Quantitative Study of Search Methods and the Effect of Constraint Salisfaction", Computer Science Dept., Carnegie Mellon University, 1984.

[3] Korf, RE., "The Complexity of Brute Force Search", Technical Report, Department of Computer Science, Columbia University, 1984.

[4] Nilsson, N., *Problem Solving Methods in Artificial Intelligence,* McGraw-Hill, 1971.

[5] Slate, D. J., and Atkin, L. Ft., "CHESS 4.5 The Northwestern University Chess Program", in *Chess Skill in Man and Machine,* P. Frey (Ed.). Springer Verlag, 1977.