CMPUT657
Projects and Presentations

November 3, 2004

1 Teams

Course presentations and projects are to be done in teams of two. Please send email to jonathan@cs with your partnership.

2 Presentation

Choose a recent journal or conference paper in heuristic search to present in class. The instructor must approve your choice of paper. You might also consider your project at the same time, since it would be useful (but is not required) to present in class a paper that is related to your course project.

For this assignment, you have two requirements:

1. You will make a 30-minute presentation in class, plus 10 minutes for questions, on your paper. Make use of the A/V equipment in the room; computer-based or overhead presentations only. The white board can only be used for supplementary material.

2. All team members must participate in the presentation. I would expect that for a two person team, each person would do roughly half the presenting.

3. Hand in a paper (5-page typeset maximum) that summarizes the paper, identifies the contribution, critiques it (identifying the weaknesses and errors in the paper), and puts the work in perspective with the related work cited in the paper.

The major sources for papers include:

1. AAAI conference (American Association for Artificial Intelligence)

2. IJCAI conference (International Joint Conference on AI)

3. Artificial Intelligence Journal – all papers are on the web at www.elsevier.com.

4. JAIR (Journal of Artificial Intelligence Research) – all papers are on the web at www.jair.org.

5. Web searches – lots of good papers can be found through web searches

If you have trouble finding a paper, see me as I may have a copy.

Contact the instructor as soon as possible to schedule your talk. First come, first served.

Your presentation will be evaluated based on:

1. Instructor's assessment

2. Class assessment

3. Paper summary – due two days after your presentation

All written material must be professionally done: spelling and grammar mistakes are not acceptable.

3 Project

Your team should choose a project. There are two options:

1. Does the thrill of building a working system turn you on? Look no further: this is the project option you have been waiting for! Implementation: Choose an interesting heuristic search project and build an implementation. The implementation must include a maximum 10-pages of double spaced documentation that discusses the design, implementation, and results of your system.

2. Tired of programming? Then this is the option for you! Here is a chance to become an expert in a sub-area of artificial intelligence! Literature review: Choose a topic in heuristic search, read the important papers in that area, and write an essay summarizing the state-of-the-art in that area. The review must be no more than 25 pages, including figures and bibliography.

All written material must be professionally done: spelling and grammar mistakes are not acceptable.

Your choice must be approved by the instructor. To make sure that the project is not too big or too small, please submit a one-page summary of the proposed work.

The project is due on the last day of classes. However, projects will be accepted up to and including December 20 with a 1% late penalty.

4 Presentation and Project Suggestions

The following are suggestions for presentations and/or projects. This list is by no means complete!

4.1 Search Techniques

1. Cache-friendly search. The open list in A* and the transposition table in IDA* and alpha-beta have performance problems; they are not cache friendly. Are there ways of reorganizing the data to get better cache performance? See, for example, [4] for a so-so attempt to solve this problem.

2. Pattern databases are useful for both single-agent search (typically used to improve the heuristic estimator) and for two-player games (typically used to replace search). Can you find and implement a novel application of pattern databases?

4.2 Single-agent Search

1. Korf and Felner created pattern databases that were additive; you could look up multiple pattern database values and *add* the results to get an admissible heuristic [17]. Felner has applied these ideas to the vertex cover and graph partitioning problems [5].

2. Many single-agent search domains do not have the luxury of searching for a few hours to get the optimal result. What do you do if you can only search for a second... and that is not enough to find the optimal (or any) answer. Real-time A* search has a time constraint put on the search. What can you do to maximize the result of a single-agent search given a tight time limitation? See, for example, Sven Koenig's work on Learning Real-Time A* [12].

3. The best planning systems today are based on heuristic search. The problem here is to take an application specified using an application-independent language and from this extract a heuristic evaluation function. Take a look at what the best planners are doing. For example, see the HSP program [2].

4. Rob Holte has been working on using pattern databases to automatically create search heuristics [9]. Can you automate building a search heuristic for an interesting domain?

5. Rob Holte and Istvan Hernadvolgyi looked to prove Korf and Reid's conjecture about the space-time tradeoff in A* [8]. Does the conjecture hold for other domains?

6. Korf and Zhang propose divide-and-conquer frontier search as an efficient algorithm for the multiple sequence alignment problem [18]. Can you verify their results? Can you apply the idea to a different domain?

7. The partial expansion A* (PEA*) algorithm attempts to reduce the space required by the A* lists [24]. Parent nodes are partially expanded and only "promising" nodes are saved; other nodes can be recomputed later on. The paper has some impressive claims, but there is evidence that

the algorithm only works well when you have "easy" problems; when the search is hard, PE* loses most of its effectiveness. Can you validate this?

8. Bidirectional search promises an exponential reduction in single-agent search effort. Unfortunately, there are problems with this idea that limit its usefulness in practice. Kaindl and Kainz provide a good overview of the state of the art [10]. Try building a bi-directional solver for an interesting domain and compare the results to A*/IDA*.

9. Endgame databases are usually done offline. What is you want to combine searching backward building the endgame databases (or search perimeter) at the same time as you are searching forward? Manzini proposed such an algorithm [20]. I would like to see this algorithm implemented and analyzed.

10. Limited discrepancy search does an unusual form of iteration [7, 13]. It assumes you have very good move ordering. The initial pass assumes the best move is best always. The next iteration assumes that along each path only one move ordering decision is wrong. The next iteration assumes that along each path two move ordering decisions are wrong.

11. Korf has applied heuristic search to a number of interesting applications: number partitioning [14], bin packing [15], and sequence alignment [19]. Choose an applications (one of the above or another) and try out one of these algorithms. I am particularly interested in seeing an implementation of Korf and Zhang's (multiple) sequence alignment algorithm.

4.3 Adversary Search

1. Korf and Chickering proposed a best-first search algorithm for two-player search [16]. Unfortunately, no one uses it in practice! Recently, Shoham and Toledo proposed a variation on this algorithm that shows some promise – especially when it comes to increasing the potential parallelism in the search [22]. Try implementing this algorithm, perhaps with Ataxx. How does it perform? Can you improve the algorithm?

2. There have been many attempts to improve the move ordering in alpha-beta search. Two new schemes are move influence [6] and neural move maps [11]. Implement them in a game-playing program. Do these papers offer hype or hope?

3. Proof number search (PN), a variation of conspiracy number search, has been shown to be effective for solving two-player games. The PN* algorithm combines proof numbers with iterative deepening. The result is a program that can solve tsume-shogi problems that are 1500 moves deep [21]! How about trying solving a game like checkers (see me for help)? Can you come up with an efficient Ataxx endgame solver?

4. Some games have more than two players. Multi-player alpha-beta is used in a turn-based game where all players are competing against each other. Pruning using multi-player alpha-beta is not as effective in the multi-player case as it is in the two-player case [23]. Build a high-performance multi-player program for a game such as Chinese checkers.

5. Buro's ProbCut [3] is an elegant idea that needs to be investigated in other search domains. How does it do in Ataxx?

6. Anshelevich proposes a powerful proof technique for analyzing positions in the game of Hex [1]. Nice ideas; can they be applied anywhere else?

4.4 Other Topics

Here is a brief selection of other project topics that might be of interest:

1. For Rush Hour puzzles, a solution to this game can be viewed as a tree where each node is a move of one car (or animal). The children of each car-move are actually a chain of moves that must be done in order to make the parent move possible. These cars are actually blocking the parent car. The challenge is to solve the problem without a forward search. The project would be to build a planner that finds a (near) optimal solution. This planner should presumably solve such problems with much larger state-space like rush-hour with 100x100 spaces and 500 cars.

2. The game of Quoridor. Rules and introduction to this game can be found in: `http://www.math.uaa.alaska.edu/~afkjm/ai_games/quoridor/quoridor.html` The problem with this game is the large branching factor due to many possible locations that a barrier can be placed. I would like to see a good game-playing engine! The challenges here would be: a) finding a good evaluation function for the game, and b) finding a suitable selective search algorithm with some kind of forward move pruning (since not all the children of a node should be examined).

3. Games of chance; how do you handle the role of the dice?

4. Temporal difference learing – can you use it to build a strong Ataxx evaluation function?

5. Constraint programming – using heuristic search to solve large problem instances.

6. Performance of proof set search versus proof number search on a DAG.

Having trouble finding a paper? How about these ideas?

1. Ginsberg's new algorithm for solving bridge hands (JAIR).

2. The *Deep Blue* article (AIJ).

3. Computer Scrabble (AIJ).

4. Building endgame databases.

5. Pathology – we did not discuss this in class, but sometimes search can lead to pathological behaviour – deeper search leads to a poorer-quality answer!

References

[1] Vadim Anshelevich. A hierarchical approach to computer hex. *Artificial Intelligence*, 134(1-2):101–120, 2002.

[2] Blai Bonnet and Hector Geffner. Planning as heuristic search. *Artificial Intelligence*, 129:5–33, 2001.

[3] Michael Buro. Probcut: An effective selective extension of the alpha-beta algorithm. *Journal of the International Computer Chess Association*, 18(2):71–76, 1995.

[4] Stefan Edelkamp and Stefan Schrodl. Localizing A*. *AAAI National Conference*, pages 885–890, 2000.

[5] Ariel Felner. *Improving Search Techniques and Using Them on Different Environments.* PhD thesis, 2001.

[6] Kieran Greer. Computer chess move-ordering schemes using move influence. *Artificial Intelligence*, pages 236–250, 2000.

[7] W. Harvery and Matt Ginsberg. Limited discrepancy search. *International Joint Conference on Artificial Intelligence*, pages 607–613, 1995.

[8] Rob Holte and Istvan Hernadvolgyi. A space-time tradeoff for memory-based heuristics. *AAAI National Conference*, pages 704–709, 1999.

[9] Rob Holte and Istvan Hernadvolgyi. Steps towards the automatic creation of search heuristics, 2001. `www.cs.ualberta.ca/~holte/CMPUT651/unpublished.ps`.

[10] Hermann Kainl and Gerhard Kainz. Bidirectional heuristic search reconsidered. *Journal of Artificial Intelligence Research*, 7:283–317, 1997.

[11] Levente Kocsis, Jos Uiterwijk, Eric Postma, and Jaap van den Herik. The neural movemap heuristic in chess. *Computers and Games*, 2002. Preprint available from Jonathan Schaeffer.

[12] Sven Koenig. Minimax real-time heuristic search. *Artificial Intelligence*, 129:165–197, 2001.

[13] Richard Korf. Improved limited discrepancy search. *AAAI National Conference*, pages 286–291, 1996.

[14] Richard Korf. A complete anytime algorithm for number partitioning. *Artificial Intelligence*, 106(2):181–203, 1998.

[15] Richard Korf. A new algorithm for optimal bin packing. *AAAI National Conference*, pages 731–736, 2002.

[16] Richard Korf and Max Chickering. Best-first minimax search. *Artificial Intelligence*, pages 299–337, 1996.

[17] Richard Korf and Ariel Felner. Disjoint pattern database heuristics. *Artificial Intelligence*, 134(1-2):9–22, 2002.

[18] Richard Korf and Weixiong Zhang. Divide-and-conquer frontier search applied to optimal sequence alignment. *AAAI National Conference*, pages 910–916, 2000.

[19] Richard Korf and Weixiong Zhang. Divide-and-conquer frontier search applied to optimal sequence alignment. *AAAI National Conference*, pages 910–916, 2000.

[20] G. Manzini. Bida*: An improved perimeter search algorithm. *Artificial Intelligence*, 75(2):347–360, 1995.

[21] Masahiro Seo, Hiroyuki Iida, and Jos Uiterwijk. The pn*-search algorithm: Application to tsume-shogi. *Artificial Intelligence*, 129:253–277, 2001.

[22] Yaron Shoham and Sivan Toledo. Parallel randomized best-first minimax search. *Artificial Intelligence*, 137:165–196, 2002.

[23] Nathan Sturtevant and Richard Korf. On pruning techniques for multi-player games. *AAAI National Conference*, pages 201–207, 2001.

[24] Takayuki Yoshizumi, Teruhisa Miura, and Toru Ishida. A* with partial expansion for large branching factor problems. *AAAI National Conference*, pages 923–929, 2000.