CS 748 (Spring 2022): Weekly Quizzes

Instructor: Shivaram Kalyanakrishnan

May 9, 2022

Note. Provide justifications/calculations/steps along with each answer to illustrate how you arrived at the answer. State your arguments clearly, in logical sequence. You will not receive credit for giving an answer without sufficient explanation.

Submission. You can either write down your answer by hand or type it in. Be sure to mention your roll number. Upload to Moodle as a pdf file.

Week 12

In a game of RBC between players 1 and 2, let I^t denote the information set (equivalently, the *board set*) maintained by Player 1 after Player 2 has just completed its t-th regular move (hence it is now Player 1's turn to perform a recon move). Let I^t_+ be the information set of Player 1 after it plays its recon move r, and let I^t_{++} be the information set of Player 1 after it subsequently plays its regular move m. Thereafter Player 2 makes its recon and regular moves, and Player 1 updates to information set I^{t+1} (and this sequence cycles).

Explain how I^t_+ is obtained from I^t, how I^t_{++} is obtained from I^t_+, and how I^{t+1} is obtained from I^t_{++}. Also describe relations between the sizes of I^t, I^t_+, I^t_{++}, and I^{t+1} under common circumstances (ignoring corner cases such as illegal moves). [4 marks]

Solution.

- When the information set is I^t, a recon move r is played to obtain I^t_+. I^t_+ is a subset of I^t containing all boards $I \in I^t$ that are consistent with the results of r. Naturally, we have $|I^t_+| \leq |I^t|$.
- When Player 1 plays a regular move m, for each board $I \in I^t_+$, the resulting "afterstate" I', obtained by playing m on I, is put into I^t_{++}. I^t_{++} does not contain any states other than these afterstates. If no two states in I^t_+ lead to the same afterstate by playing m, we will have $|I^t_{++}| = |I^t_+|$.

  Now, it might occasionally happen that the move m selected by Player 1 is not the one played by the engine, since m might be impossible to play on the true underlying board. Instead the engine may play move $\bar{m}$, which could be different from m. Moreover, the context is typically such that if $\bar{m} \neq m$, the player obtains some form of feedback (such as "illegal move", or "piece capture at a given position"). This feedback may help prune I^t_+ to a smaller set, on which the preceding argument should be invoked with $\bar{m}$ in place of m. In such cases, we are likely to encounter $|I^t_{++}| \leq |I^t_+|$.
- In principle, Player 2 can play any possible move m' from the afterstate reached after Player 1's move m (or $\bar{m}$). Hence, for every afterstate $I \in I^t_{++}$, we generate an entire set $\text{Next}(I)$ of possible next states (from Player 1's perspective) that can be reached from I using *some* move m' by Player 2. Thereafter we set $I^{t+1} = \cup_{I \in I^t_{++}} \text{Next}(I)$. Consequently we have $|I^{t+1}| \geq |I^t_{++}|$.

Week 11

In the paper listed below (also briefly mentioned in the week's lecture), Silver and Veness describe the POMCP algorithm for POMDPs.

> Monte-Carlo Planning in Large POMDPs
> David Silver and Joel Veness.
> Advances in Neural Information Processing Systems 23, 2164–2172, Curran Associates, 2010.

a. What is the main difference between POMCP on the one hand, and traditional POMDP planning algorithms such as value iteration and policy iteration on the other? On a related note, describe the properties of tasks on which POMCP would be a better choice than the traditional algorithms, and vice versa. [2 marks]

b. Provide a summary of POMCP, indicating the main steps that it performs. [2 marks]

Solution.

a. POMCP is an algorithm for *decision time planning* in POMDPs, much in the spirit of the UCT algorithm for MDPs. By contrast, value iteration and policy iteration are planning algorithms that are run off-line to compute (approximately) optimal policies for POMDPs, which can be looked up very efficiently when deployed for action selection. POMCP could be a good choice when, indeed, the agent has sufficient time for planning its action at each step.

b. Like any POMDP algorithm, POMCP has to (1) select an action from the current belief state, and (2) depending on the observation received, update to the next belief state. POMCP performs (1) by running a version of the UCT algorithm, which samples states from the current belief state and performs roll-outs from them. Interestingly, the belief state is maintained as a set of particles (states), and the belief update in (2) is implemented using these particles.

Week 10

Question. Answer the following questions based on the week's lecture.

a. Consider the POMDP described on Slide 2 of the lecture, for which we work out a belief update on Slide 9. Suppose the initial belief state is $b_0 = (\frac{1}{2}, \frac{1}{4}, 0, \frac{1}{4})$, the agent takes action **L**, and consequently observes NO-GOAL. Compute b_1. [3 marks]

b. What is the practical difficulty in performing policy iteration on POMDPs? [1 mark]

Solution.

a.

$$b_1(1) \propto b_0(1) \times 0.9 + b_0(2) \times 0.9 + b_0(3) \times 0 + b_0(4) \times 0 = \frac{9}{20} + \frac{9}{40} = \frac{27}{40}.$$
$$b_1(2) \propto b_0(1) \times 0.1 + b_0(2) \times 0 + b_0(3) \times 0 + b_0(4) \times 0 = \frac{1}{20}.$$
$$b_1(3) \propto 0.$$
$$b_1(4) \propto b_0(1) \times 0 + b_0(2) \times 0 + b_0(3) \times 0 + b_0(4) \times 0.1 = \frac{1}{40}.$$

Consequently we get $b_1 = (\frac{9}{10}, \frac{1}{15}, 0, \frac{1}{30})$.

b. The idea behind policy iteration is that with every iteration, we obtain a dominating policy. In MDPs, a policy can be represented as a table that specifies an action for each state. In POMDPs, policies instead map belief states to actions, and hence need a different representation, such as a finite state controller whose "states" correspond to regions of the belief space (much larger in number than the number of states). When proceeding from policy π to dominating policy π', the size of the underlying representation invariably increases (and quite significantly). Hence, in practice, improving policies cannot be computed beyond 5–10 iterations.

Week 9

Question. Answer the following questions based on the paper by Jain *et al.* on the use of PPR Martingale confidence sequences for PAC mode estimation.

a. What is the main difference between the PAC mode estimation problem and the PAC subset selection problem in bandits (covered in Week 8)? [1 mark]

b. The SPRT rule applied to the blockchain application makes use of $f_{\max}$, which is an upper bound on the true Byzantine fraction f. Now suppose that you are given $f_{\max}$ and the assurance it upper-bounds f, can you use this information to improve the PPR-1v1 stopping rule (which in its original form is independent of $f_{\max}$)? You will obtain partial marks for qualitative reasoning; to obtain full marks work all the way to a concrete stopping rule based on PPR Martingale confidence sequences that uses $f_{\max}$. [3 marks]

c. Jain *et al.* introduce a rule called "PPR-MD" in addition to PPR-1vr and PPR-1v1. What are the two main shortcomings of PPR-MD when compared with PPR-1v1? [2 marks]

Solution.

a. In mode estimation, the algorithm is a *stopping rule*: it only has to decide when to stop (and δ-correctly declare the mode). In bandit subset selection, the algorithm must not only decide when to stop (and δ-correctly declare the optimal arm), it has the additional job of deciding which arms to sample (based on the history).

b. In PPR-1v1, by default we take a uniform prior over $[0,1]$ on the mean of the unknown Bernoulli variable. Given the additional information that the mean is lower-bounded by $1-f_{\max}$, we can gain some advantage by assuming an alternative prior that is uniform over the region $F = F_{\text{low}} \cup F_{\text{high}}$, where $F_{\text{low}} = [0, f_{\max}]$ and $F_{\text{high}} = [1-f_{\max}, 1]$. For $x \in F$, we therefore have

$$\pi^0(x) = \frac{1}{2f_{\max}},$$

with the update from time step t to $t+1$ depending on the class of the t-th sample, as below.

$$\pi_x^{t+1} \propto \begin{cases} \pi^t(x) \cdot x & \text{class of } t\text{-th sample} = 1 \\ \pi^t(x) \cdot (1-x) & \text{class of } t\text{-th sample} = 0, \end{cases}$$

where there are s occurrences of class 1 and f occurrences of class 0. It follows that

$$\begin{aligned} \pi^t(x) &= \frac{\frac{1}{2f_{\max}} x^s (1-x)^f}{\int_{y \in F} \frac{1}{2f_{\max}} y^s (1-y)^f dy} \\ &= \frac{\text{Beta}(x; s+1, f+1)}{1 - \text{BetaCDF}(1-f_{\max}; s+1, f+1) + \text{BetaCDF}(f_{\max}; s+1, f+1)}, \end{aligned}$$

and consequently the PPR for $x \in F$ is

$$\text{PPR}^t(x) = \frac{\pi^0(x)}{\pi^t(x)} = \frac{1 - \text{BetaCDF}(1-f_{\max}; s+1, f+1) + \text{BetaCDF}(f_{\max}; s+1, f+1)}{2f_{\max}\text{Beta}(x; s+1, f+1)}.$$

Moreover, with the knowledge that the true mean must lie in F, our confidence set only needs to be $\{x \in F : PPR(x) < \frac{1}{\delta}\}$. We can stop as soon as the confident set stops intersecting either F_{low} (hence declaring 1 as the mode) or it stops intersecting F_{high} (hence declaring 0 as the mode).

c. The PPR-MD stopping rule requires the solution of an optimisation problem, which is much more computationally expensive than PPR-1v1 and PPR-1vr. Moreover, the authors have shown that PPR-1v1 necessarily terminates no later than PPR-MD on every sample trajectory. In short, PPR-1v1 is preferable to PPR-MD both in terms of computational complexity and sample complexity.

Week 8

Question. This question requires you to modify LUCB to solve a different problem from subset selection. Suppose, as usual, that your algorithm interacts with a multi-armed bandit instance in which the rewards are all Bernoulli. The arms' means are $p_1, p_2, \ldots, p_n$, satisfying $p_1 > p_2 > \cdots > p_n$, where $n \geq 3$ is an *odd number*. . Your algorithm must sample the arms to identify the *median* arm: that is, arm $m \stackrel{\text{def}}{=} \frac{n+1}{2}$, correctly with probability at least $1 - \delta$.

The idea is that you should use upper and lower confidence bounds on the means of each arm, just like the Hoeffding's Inequality-based ones applied in LUCB. However, you should revise both the stopping rule and the sampling strategy so as to identify the median arm correctly with probability at least $1 - \delta$, while obtaining an upper bound on the sample complexity that scales with the "hardness" of the bandit instance. You do not need to provide rigorous proofs—rough sketches that refer to steps of the LUCB proof will suffice.

Provide the intuition behind your stopping rule, explaining how it ensures δ-correctness at termination. For the sampling strategy you propose, what hardness quantity H do you think will emerge? [5 marks]

Solution. We can draw exactly the same lower and upper confidence bounds on the means of each arm as done in LUCB. As shown for LUCB, the probability that any true mean ever exceeds its LCB or UCB (in the appropriate direction) is at most δ. Now, at time t, let the sets A^t, B^t, and C^t partition the set of arms A such that

1. $|A^t| = m - 1, |B^t| = 1, |C^t| = m - 1$, and
2. for $(a, b, c) \in A^t \times B^t \times C^t, \hat{p}_a^t \geq \hat{p}_b^t \geq \hat{p}_c^t$.

We stop and return the single element $b \in B^t$ when b's UCB falls below the least LCB from A^t, and b's LCB exceeds the highest UCB from C^t. From the δ-correctness of the confidence bounds, it would follow that the true mean of b is exceeded by the true means of all $m - 1$ arms in A^t, and the true mean of b exceeds the true means of all $m-1$ arms in B^t, hence making b the median arm.

As for the sampling strategy, the most natural approach would be to sample, at each step t, the arm in A^t with the least LCB, arm $b \in B^t$, and the arm in C^t with the highest UCB. The hardness quantity resulting from such a strategy would vary as $H = \sum_{i=1}^{m-1} \frac{1}{(p_i - p_m)^2} + \sum_{i=m+1}^{n} \frac{1}{(p_m - p_i)^2}$.

Week 7

Question. Both questions below relate to the structure of the policy space for MDPs. Concretely consider an MDP $M = (S, A, T, R, \gamma)$, with notations and assumptions as usual. Let $|S| = n \geq 2$ and $|A| = k \geq 2$.

a. For each policy π for M, let the *value* of π be the scalar $V(\pi) \stackrel{\text{def}}{=} \sum_{s \in S} V^\pi(s)$. Let Π be the set of all (k^n) deterministic policies for M.

   Assume that no two distinct policies in Π have the same value—hence policies in Π are totally-ordered based on their values. Let $\pi^\star \in \Pi$ denote the optimal policy (which clearly has the highest value) and let $\pi_2 \in \Pi$ be the policy in Π with the *second highest* value. In other words, every policy in $\Pi \setminus (\{\pi^\star\} \cup \{\pi_2\})$ has a lower value than that of π_2.

   You are given M and $\pi^\star$ as input. Provide a procedure to compute π_2 using $\text{poly}(n, k)$ arithmetic operations. Establish the correctness of your procedure. [3 marks]

b. In this part, you are asked to prove a generalisation of the policy improvement theorem presented in CS 747, in particular generalising it from deterministic to stochastic policies. Note that the set of all stochastic policies contains the set of deterministic policies Π.

   Let π and π' be *distinct*, *stochastic* policies for M, which satisfy the following condition: for $s \in S, a \in A$,

$$Q^\pi(s,a) > V^\pi(s) \implies \pi'(s,a) \geq \pi(s,a),$$
$$Q^\pi(s,a) \leq V^\pi(s) \implies \pi'(s,a) \leq \pi(s,a).$$

   Prove that if π is not an optimal policy, then $\pi' \succ \pi$. You may state and use results already proven in CS 747. [3 marks]

Solution.

a. Our solution is built upon the claim that π_2 differs from $\pi^\star$ in exactly one state.

   To see why, consider any non-optimal policy π. Since π will have one or more improvable states, and switching any subset of improvable states to improving actions guarantees policy improvement, we can obtain a dominating policy π' by switching to an improving action in exactly one improvable state of π. Thus, if π is a non-optimal policy, there exists a dominating policy π' that differs from π in exactly one state.

   Now, suppose π_2, which is non-optimal, differs from $\pi^\star$ in two or more states. By the argument provided above, we can obtain a policy π_2' that dominates π_2, and differs from it in exactly one state. Since $\pi^\star$ differs from π_2 in more than one state, and π_2' differs from π_2 in exactly one state, it is clear that $\pi_2' \neq \pi^\star$. However, the existence of such a π_2', whose value must lie between those of π_2 and $\pi^\star$, contradicts the fact that π_2 has the second highest value among all policies. Hence, we are assured that π_2 can differ from $\pi^\star$ in exactly one state.

   If we are given $\pi^\star$, we can evaluate all $n(k-1)$ "one-action-different" neighbours of $\pi^\star$ and identify the one with the highest value—it is going to be π_2. Since policy evaluation and the computation of policy values all need $\text{poly}(n, k)$ arithmetic operations, our overall computation is also $\text{poly}(n, k)$.

b. The following working establishes that $B^{\pi'}(V^\pi) \succeq V^\pi$: for each $s \in S$, it shows that $B^{\pi'}(V^\pi)(s) - V^\pi(s)$ is a sum of non-negative terms. For $s \in S$:

$$\begin{aligned}
B^{\pi'}(V^\pi)(s) - V^\pi(s) &= \sum_{a \in A} \pi'(s,a) \sum_{s' \in S} T(s,a,s')\{R(s,a,s') + \gamma V^\pi(s')\} - V^\pi(s) \\
&= \sum_{a \in A} \pi'(s,a) Q^\pi(s,a) - V^\pi(s) \\
&= \sum_{a \in A} \pi'(s,a)(Q^\pi(s,a) - V^\pi(s)) \\
&= \sum_{a \in A} \big(\pi'(s,a) - \pi(s,a)\big)\,(Q^\pi(s,a) - V^\pi(s)) + \sum_{a \in A} \pi(s,a) Q^\pi(s,a) - \sum_{a \in A} \pi(s,a) V^\pi(s) \\
&= \sum_{a \in A} \big(\pi'(s,a) - \pi(s,a)\big)\,(Q^\pi(s,a) - V^\pi(s)) + V^\pi(s) - V^\pi(s) \\
&= \sum_{a \in A} \big(\pi'(s,a) - \pi(s,a)\big)\,(Q^\pi(s,a) - V^\pi(s))\,.
\end{aligned}$$

By repeatedly applying $B^{\pi'}$, we gather that $V^{\pi'} \succeq V^\pi$: that is, $\pi' \succeq \pi$. To show $\pi' \succ \pi$, we require an additional assumption that was not made explicit in the question—we need that for some *improvable* state s, $\pi'(s) > \pi(s)$ (the question assures only $\geq$, which is inadequate). With this assumption, we have that $B^{\pi'}(V^\pi) \succ V^\pi$, which leads to $\pi' \succ \pi$.

Week 6

Question. One of the central principles in human computation is the *wisdom of the crowd*: that by suitably aggregating individuals' responses, we can obtain results that are more accurate or useful than the average response. Answer questions a and b after going through the three papers listed below.

> Whose Vote Should Count More: Optimal Integration of Labels from Labelers of Unknown Expertise
> Jacob Whitehill, Paul Ruvolo, Tingfan Wu, Jacob Bergsma, and Javier Movellan.
> Advances in Neural Information Processing Systems 22, 2035–2043, Curran Associates, 2009.
>
> Intuitive Biases in Choice versus Estimation: Implications for the Wisdom of Crowds
> Joseph P. Simmons, Leif D. Nelson, Jeff Galak, and Shane Frederick.
> Journal of Consumer Research, 38(1):1–15, 2011.
>
> Studying the "Wisdom of Crowds" at Scale
> Camelia Simoiu, Chiraag Sumanth, Alok Mysore, and Sharad Goel.
> In Proceedings of the Seventh AAAI Conference on Human Computation and Crowdsourcing (HCOMP-19), pp. 171–179, AAAI Press, 2019.

a. What is the specific aspect in harnessing the wisdom of the crowd that Whitehill *et al.* address? What is their methodological contribution? [2 marks]

b. The papers by Simmons *et al.* and Simoiu *et al.* both provide at least one result contrary to the wisdom-of-the-crowd principle, wherein a reasonable aggregation of individuals' responses results in an *inferior* outcome. Briefly describe a negative result from each paper. [2 marks]

Solution.

a. Whitehill *et al.* propose a relatively sophisticated approach for *aggregating* the labels provided by human labelers for an image labeling task. Whereas a naïve aggregation strategy would be to consider the median or some other simple statistic as the label for each image, Whitehill *et al.* propose a graphical model that can ultimately provide more accurate labels. The graphical model simultaneously assigns probabilities to image labels, the proficiency of individual labelers, and the difficulty level of each image. Driven by the data, the model can therefore up-weight the labels of expert labelers, for example—thereby increasing the overall labeling accuracy.

b. Simmons *et al.* document the presence of systematic bias in a betting market for games: even when provided knowledge of the "true" odds, people tend to bet disproportionately for the favourites. Simoiu *et al.* provide a stark illustration of *social influence* hindering the wisdom of the crowds. They observe that when the "consensus" of previous responses is given as an input to new responders, often wrong answers get reinforced rather than corrected, hence resulting in an overall drop in accuracy.

Week 5

Question. The following article presents an empirical comparison of Sarsa (a TD learning algorithm) and NEAT (an evolutionary algorithm).

> Critical factors in the empirical performance of temporal difference and evolutionary methods for reinforcement learning
> Shimon Whiteson, Matthew E. Taylor, Peter Stone.
> Autonomous Agents and Multi-Agent Systems, 21(1):1–35, 2010.

To perform their study, the authors examine how different properties of the task at hand influence the performance of each algorithm. What are the two main task properties they study? How is the effect of each of these properties determined (that is, what are the supporting experiments), and what are the findings? Do you agree with the conclusions? [4 marks]

Solution. The authors consider the effect of two task properties—the amount of state aliasing and the amount of stochasticity—in determining the performance of the two learning algorithms. The authors synthetically create variants of tasks so as to isolate the effect of task properties: thus, for example, if T is the original task, a new task T' is created by introducing state aliasing into T (or removing state aliasing from T). Evaluating the relative performance of the methods on T and T' then sheds light on the effect of state aliasing.

The authors pick T from among Mountain Car and Keepaway, and for each choice of T, construct variants T' and T'' that differ from T along one of the task properties studied. The authors find evidence that Sarsa is affected more by state noise and less by stochasticity (or action noise) compared to NEAT. Since Sarsa relies more heavily than NEAT on the assumption that state transitions are Markovian, it is to be expected that Sarsa's performance degrades relatively more rapidly due to state aliasing (which invalidates the Markov assumption) and less due to action noise (which does not affect the Markovian assumption). On the other hand, action noise increases the variance in fitness evaluations, thereby increasing the chances that less fit parents will get selected under NEAT to propagate genes to future generations.

Week 3

Question.

a. In both papers you have read this week—for playing Atari games (Mnih *et al.*, 2015) and for the game of Go (Silver *et al.*, 2016)—the underlying neural network passes the input through *convolutional* layers. Why are convolutional neural networks a good choice for both these applications? [2 marks]

b. Whereas the agents trained by Mnih *et al.* (2015) are eventually able to exceed human-level performance on the vast majority of Atari games, they do not do so on a handful of games. See Figure 3 in their paper: one of the unsuccessful games for the agent is Ms. Pac-Man. Why do you think the DQN algorithm proposed by Mnih *et al.* is unable to perform well on Ms. Pac-Man? [1 mark]

c. What is the main difference between the result achieved by Silver *et al.* (2016) and that described in the paper below by Schaeffer *et al.* (2007) (other than the obvious one that they are about different games)? [1 mark]

Checkers is Solved
Jonathan Schaeffer, Neil Burch, Yngvi Björnsson, Akihiro Kishimoto, Martin Müller, Robert Lake, Paul Lu, and Steve Sutphen.
Science, 317: 1518–1522, 2007.

Solution.

a. In both applications, the input is an image (or sequence of images). Convolutional layers in a neural network (typically the first few) apply "filters" on contiguous patches of the input, and are effective at identifying spatially-coherent patterns, such as some type of object in an Atari game, or a cluster of black stones in Go. One may expect that the features abstracted out through convolution convey information that is directly useful for decision making, such as whether the agent is due for a collision in the Atari game, or whether a move would lead to a capture in Go. It is remarkable that the entire mapping from input to actions, which involves the convolutional operators themselves, is learned end-to-end using RL in both applications.

b. For playing Ms. Pac-Man successfully, it would appear necessary to *plan* strategies such as "reaching a food pill" or "escaping through a hole"—and execute such plans by stringing together the appropriate low-level actions (moving north, south, east, and wast). However, the neural network used for learning by Mnih *et al.* is a simple feed-forward mapping from the input images to the low-level actions. Without any constructs in place to encode and persist long-term actions (such as a memory with historical actions), and without explicit decision-time planning (to simulate the future), it is hard for the agent to learn/execute effective temporally-extended action sequences. Most of the other Atari games included in the tests can be tackled effectively with the direct associative mappings encoded by the neural network.

c. Schaeffer *et al.* provide a minimax-optimal strategy from the starting position for the game of Checkers, along with a proof of minimax-optimality. As the product of largescale computation, they furnish a strategy π_1 that guarantees at least a draw for the first player, and also a strategy π_2 for the second player that forces a draw against π_1. By contrast, AlphaGo

and variants, while they may be considered "very good" or even "unbeatable" by *human* standards, currently have no proof of optimality. In other words, we do not know if there is a program that can defeat AlphaGo. Given the size of its game tree, it seems impractical at the moment to construct a provable minimax-optimal strategy for Go.

Week 2

Question. The textbook by Poole and Mackworth (2017) has a brief mention of **pattern databases**. Read the introduction (Section 1) in this article by Felner *et al.* (2007) to get a better understanding of pattern databases.

> Compressed Pattern Databases
> Ariel Felner, Richard E. Korf, Ram Meshulam, Robert Holte.
> Journal of Artificial Intelligence Research, 30: 213–247, 2007.

a. What are pattern databases? What is their connection with admissible heuristics? [2 marks]

b. What particular challenge related to pattern databases is addressed by Felner *et al.*? How is the challenge addressed? [2 marks]

Solution.

a. It is desirable for heuristics to be admissible, since they guarantee (say in conjunction with a search algorithm such as A*) that an optimal solution will be found. However, the time taken for the search depends on the actual value given by the heuristic to different nodes: the higher the value, the more efficient the search procedure. Now, it is non-trivial in general to come up with heuristics that give large values to nodes, while simultaneously satisfying the admissibility constraint of the values being upper-bounded by the optimal cost to goal. Pattern databases are a solution to this problem. To obtain heuristic value $h(n)$ for node n, a less-constrained variant (a simplification) of the original search task is solved starting from n, and the optimal cost to goal from n on this simpler task is taken as $h(n)$. Since the new task is simpler, it is guaranteed that the cost to goal on the original task will not be less than $h(n)$. Often, multiple nodes for the original task will map to a single node for the simpler task, so it suffices to maintain a table mapping node n to heuristic value $h(n)$ for a small set of nodes. This table is called a pattern database.

b. Pattern databases can often become so large as to not fit in memory. Consequently, using them while searching can become computationally expensive. Felner *et al.* propose several techniques to compress pattern databases so they can be stored in memory and hence accessed quickly. The authors report speed-up of the search procedure by several orders of magnitude on a variety of search tasks.