Dynamic Programming for Beat Tracking

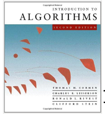
Sevag Hanssian

MUMT 621, Winter 2021

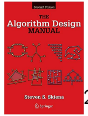
Feburary 23, 2021

Dynamic Programming for Beat Tracking
Sevag Hanssian
MUMT 621, Winter 2021
Feburary 23, 2021

Dynamic programming



1



2

Dynamic programming applies to optimization problems in which a set of choices must be made in order to arrive at an optimal solution – we seek to find a solution that **maximizes or minimizes some function**.

Greedy algorithms make the best local choice – efficient but doesn't guarantee a globally optimal solution. Exhaustive search algorithms try all combinations of choices, at a **prohibitive cost in time complexity**.

As choices are made, **subproblems** of the same form arise, often more than once. Dynamic programming's **key technique is to store solutions to each subproblem** while searching all possibilities.

1. Thomas H. Cormen et al. 2001. *Introduction to Algorithms.*
2. Steven S. Skiena. 2008. *The Algorithm Design Manual.* London. https://doi.org/10.1007/978-1-84800-070-4.

Dynamic programming

Dynamic programming applies to optimization problems in which a set of choices must be made in order to arrive at an optimal solution – we seek to find a solution that **maximizes or minimizes some function**.

Greedy algorithms make the best local choice – efficient but doesn't guarantee a globally optimal solution. Exhaustive search algorithms try all combinations of choices, at a **prohibitive cost in time complexity**.

As choices are made, **subproblems** of the same form arise, often more than once. Dynamic programming's **key technique is to store solutions to each subproblem** while searching all possibilities.

1. Thomas H. Cormen et al. 2001. *Introduction to Algorithms.*
2. Steven S. Skiena. 2008. *The Algorithm Design Manual.* London. https://doi.org/10.1007/978-1-84800-070-4.

Time complexity and Big-Oh notation

Algorithms in computer science are often described by their time complexity in a hypothetical computer where[3] **simple operations take 1 time step**. Big-Oh provides the upper bound of algorithm running time in relation to number of input elements.

```
function linearSearch(animalToFind, arrayOfAnimals) -> bool
    for every animal in arrayOfAnimals
        if animal == animalToFind
            return true
    return false
```

animals = string[5]

"cat"	"dog"	"rat"	"pig"	"fish"

animals = string[500]

"cat"	...	"fish"

Linear search is $O(n)$ – directly proportional to input size

3. Skiena 2008.

Time complexity and Big-Oh notation
Algorithms in computer science are often described by their time complexity in a hypothetical computer where[3] **simple operations take 1 time step**. Big-Oh provides the upper bound of algorithm running time in relation to number of input elements.
function linearSearch(animalToFind, arrayOfAnimals) -> bool
for every animal in arrayOfAnimals
if animal == animalToFind
return true
return false
animals = string[5]
"cat" "dog" "rat" "pig" "fish"
animals = string[500]
"cat" ... "fish"
Linear search is $O(n)$ – directly proportional to input size
3. Skiena 2008.

- Suppose an algorithm written in C is two times faster than the same algorithm written in Java – doesn't tell us much about the algorithm at all
- Ignores specific implementation details such as: programming languages, CPU models, types of storage – the goal is to understand and study algorithms in a language- and machine-independent manner. Details are less important than the worst-case time complexity

Common values for Big-Oh

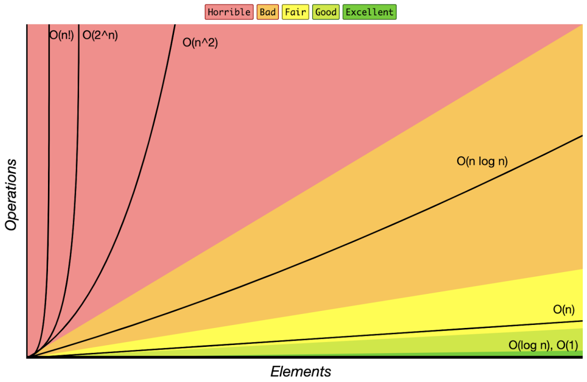
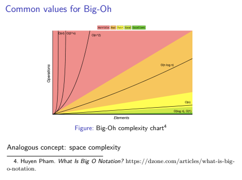


Figure: Big-Oh complexity chart[4]

Analogous concept: space complexity

4. Huyen Pham. *What Is Big O Notation?* https://dzone.com/articles/what-is-big-o-notation.



Recursion

Many algorithms are recursive in structure – to solve a problem, they call themselves recursively to deal with closely related **subproblems**[5]

Typical paradigm is **divide-and-conquer**:

1. **Divide** the problem into a number of subproblems
2. **Conquer** the subproblems by solving them recursively – if the subproblem is "small enough," just solve it in a straightforward manner
3. **Combine** the solutions to the subproblems into a solution for the original problem

5. Cormen et al. 2001.

Recursion

Many algorithms are recursive in structure – to solve a problem, they call themselves recursively to deal with closely related **subproblems**[5]

Typical paradigm is **divide-and-conquer**:

1. **Divide** the problem into a number of subproblems
2. **Conquer** the subproblems by solving them recursively – if the subproblem is "small enough," just solve it in a straightforward manner
3. **Combine** the solutions to the subproblems into a solution for the original problem

5. Cormen et al. 2001.

Recursion example – Fibonacci sequence

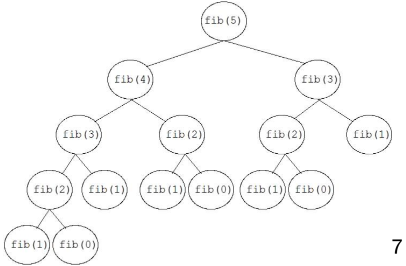
The Fibonacci sequence: 0, 1, 1, 2, 3, 5, 8, 13, ...[6]

Recursive definition:

1. $f_0 = 0$
2. $f_1 = 1$
3. $f_n = f_{n-1} + f_{n-2}$ for $n \geq 2$

```
function fib(n int) -> int {
    if n <= 1 {
return n
    }
    return fib(n-
1) + fib(n-2)
}
```

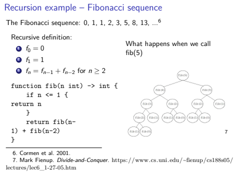
What happens when we call fib(5)



7

6. Cormen et al. 2001.

7. Mark Fienup. *Divide-and-Conquer.* https://www.cs.uni.edu/~fienup/cs188s05/lectures/lec6_1-27-05.htm



Fibonacci sequence with dynamic programming

Trade off space for time by storing the computed subproblems:

```
int[10000] cache; // global storage array

function init_fib() {
    cache[0] = 0;
    cache[1] = 1;
    for i = 2; i < 10000; i++ {
        cache[i] = -1;
    }
}
function fib(n int) -> int {
    if cache[n] == -1 {
        cache[n] = fib(n-1) + fib(n-2);
    }
    return cache[n];
}
```


Fibonacci sequence with dynamic programming
Trade off space for time by storing the computed subproblems:

```
int[10000] cache; // global storage array

function init_fib() {
    cache[0] = 0;
    cache[1] = 1;
    for i = 2; i < 10000; i++ {
        cache[i] = -1;
    }
}
function fib(n int) -> int {
    if cache[n] == -1 {
        cache[n] = fib(n-1) + fib(n-2);
    }
    return cache[n];
}
```


- No optimizing here – simply caching a recursive function gets you, most of the way to the benefits of dynamic programming
- Labmate Nestor got an improvement of 34% by caching partial results in a Python piece of music code doing something with chord keys and voicings and mus21

Beat tracking as an optimization problem

Beat tracking as an optimization problem[8]:
$C(\{t_i\}) = \sum_{i=1}^{N} O(t_i) + \alpha \sum_{i=2}^{N} F(t_i - t_{i-1}, \tau_p)$

- Compute an onset strength envelope $O(t)$ for the whole signal
- Compute a target tempo by applying autocorrelation (including some human tempo preference) to find periodicity in the onsets
- Optimize the score $C(t_i)$ of a beat at time t_i based on two terms:
  1. $O(t_i)$ – onset strength should be strong at tentative beat location t_i
  2. $F(t_i - t_{i-1}, \tau_p)$ or $F(\Delta t, \tau_p)$ – there should be consistency between the inter-beat interval Δt and the beat spacing τ_p from target tempo

  α is the weighting term to balance the importance of the 2 terms. This results in an exponential search over all time t_i[9]

8. Daniel Ellis. 2007. "Beat Tracking by Dynamic Programming." *Journal of New Music Research* 36 (March): 51–60. https://doi.org/10.1080/09298210701653344. http://www.music.mcgill.ca/~ich/classes/mumt621_09/presentations/wingate/27406228.pdf.

9. Daniel Ellis. 2013. "Lecture 10: Beat Tracking." *ELEN E4896 Music Signal Processing* (April). https://www.ee.columbia.edu/~dpwe/e4896/lectures/E4896-L10.pdf.

Beat tracking as an optimization problem
Beat tracking as an optimization problem[8]:
$C(\{t_i\}) = \sum_{i=1}^{N} O(t_i) + \alpha \sum_{i=2}^{N} F(t_i - t_{i-1}, \tau_p)$
- Compute an onset strength envelope $O(t)$ for the whole signal
- Compute a target tempo by applying autocorrelation (including some human tempo preference) to find periodicity in the onsets
- Optimize the score $C(t_i)$ of a beat at time t_i based on two terms:
  1. $O(t_i)$ – onset strength should be strong at tentative beat location t_i
  2. $F(t_i - t_{i-1}, \tau_p)$ or $F(\Delta t, \tau_p)$ – there should be consistency between the inter-beat interval Δt and the beat spacing τ_p from target tempo

  α is the weighting term to balance the importance of the 2 terms. This results in an exponential search over all time t_i[9]
8. Daniel Ellis. 2007. "Beat Tracking by Dynamic Programming." *Journal of New Music Research* 36 (March): 51–60. https://doi.org/10.1080/09298210701653344. http://www.music.mcgill.ca/~ich/classes/mumt621_09/presentations/wingate/27406228.pdf.
9. Daniel Ellis. 2013. "Lecture 10: Beat Tracking." *ELEN E4896 Music Signal Processing* (April). https://www.ee.columbia.edu/~dpwe/e4896/lectures/E4896-L10.pdf.

- in the paper he says 120bpm is a natural choice for listeners
- Exhaustive search algorithms try all combinations of choices, at a **prohibitive cost in time complexity**

Onset strength envelope and tempo estimation

Onsets mark the start of a musical note or acoustic event[10]

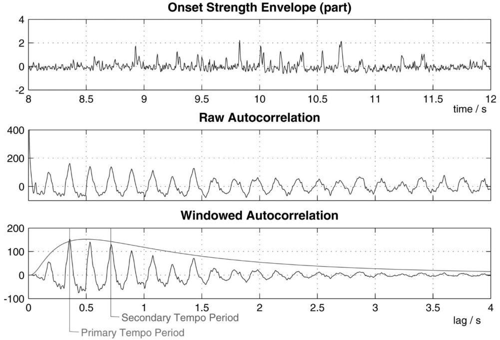
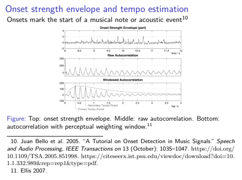


Figure: Top: onset strength envelope. Middle: raw autocorrelation. Bottom: autocorrelation with perceptual weighting window.[11]

10. Juan Bello et al. 2005. "A Tutorial on Onset Detection in Music Signals." *Speech and Audio Processing, IEEE Transactions on* 13 (October): 1035–1047. https://doi.org/10.1109/TSA.2005.851998. https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.332.989&rep=rep1&type=pdf.

11. Ellis 2007.



- Human tempo perception is known to have a bias towards 120 BPM. We apply a perceptual weighting window to the raw autocorrelation to downweight periodicity peaks far from this bias
- This plot contains all the information you need for a beat tracker! but remember the optimization goals

Beat tracking with dynamic programming

Recall: beat tracking as an optimization problem[12]:

$$C(\{t_i\}) = \sum_{i=1}^{N} O(t_i) + \alpha \sum_{i=2}^{N} F(t_i - t_{i-1}, \tau_p)$$

Recursive definition from best possible score $C^*(t)$ at time t:

$$C^*(t) = O(t) + \max_{\tau=0\ldots t}\{\alpha F(t-\tau, \tau_p) + C^*(\tau)\}$$

Record preceding beat time that gave best score:

$$P^*(t) = \mathrm{argmax}_{\tau=0\ldots t}\{\alpha F(t-\tau, \tau_p) + C^*(\tau)\}$$

The best score C^* for time t is the local onset strength, plus the best score to the preceding beat time τ that maximizes the sum of that best score and the transition cost from that time

12. Ellis 2007.

Beat tracking with dynamic programming

Recall: beat tracking as an optimization problem[12]:

$$C(\{t_i\}) = \sum_{i=1}^{N} O(t_i) + \alpha \sum_{i=2}^{N} F(t_i - t_{i-1}, \tau_p)$$

Recursive definition from best possible score $C^*(t)$ at time t:

$$C^*(t) = O(t) + \max_{\tau=0\ldots t}\{\alpha F(t-\tau, \tau_p) + C^*(\tau)\}$$

Record preceding beat time that gave best score:

$$P^*(t) = \mathrm{argmax}_{\tau=0\ldots t}\{\alpha F(t-\tau, \tau_p) + C^*(\tau)\}$$

The best score C^* for time t is the local onset strength, plus the best score to the preceding beat time τ that maximizes the sum of that best score and the transition cost from that time

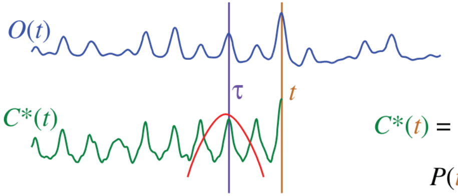
12. Ellis 2007.

Penalty term

Function F is a squared-error applied to the log-ratio of actual to ideal time spacing:

$$F(\Delta t, \tau) = -\left(\log \frac{\Delta t}{\tau}\right)^2$$

In practice, only need to search a limited range of τ since the penalty term F grows the further you are from τ_p – search in $\tau = t - 2\tau_p ... t - \frac{\tau_p}{2}$



$$C^*(t) = O(t) + \max_{\tau}\{\alpha F(t - \tau, \tau_p) + C^*(\tau)\}$$
$$P(t) = \operatorname*{argmax}_{\tau}\{\alpha F(t - \tau, \tau_p) + C^*(\tau)\}$$

Figure: Beat tracking by dynamic programming[13]

13. Ellis 2013.

Penalty term
Function F is a squared-error applied to the log-ratio of actual to ideal time spacing:
In practice, only need to search a limited range of τ since the penalty term F grows the further you are from τ_p – search in $\tau = t - 2\tau_p ... t - \frac{\tau_p}{2}$
Figure: Beat tracking by dynamic programming[13]
13. Ellis 2013.

Beat tracking with dynamic programming

- To find the set of beat times that optimize the objective function for a given onset envelope, start by calculating C^* and P^* for every time starting from zero
- Look for the largest value of C^* (which will typically be within τ_p of the end of the time range)
- This forms the final beat instant t_N where N, the total number of beats, is still unknown
- Then "backtrace" via P^*, finding the preceding beat time $t_{N-1} = P^*(t_N)$, and progressively working backwards until we reach the beginning of the signal
- Gives the entire optimal beat sequence ${t_i}^*$

Beat tracking with dynamic programming
- To find the set of beat times that optimize the objective function for a given onset envelope, start by calculating C^* and P^* for every time starting from zero
- Look for the largest value of C^* (which will typically be within τ_p of the end of the time range)
- This forms the final beat instant t_N where N, the total number of beats, is still unknown
- Then "backtrace" via P^*, finding the preceding beat time $t_{N-1} = P^*(t_N)$, and progressively working backwards until we reach the beginning of the signal
- Gives the entire optimal beat sequence ${t_i}^*$

- Thanks to dynamic programming, we have effectively searched the entire exponentially-sized set of all possible time sequences in a linear-time operation.
- This was possible because, if a best-scoring beat sequence includes a time t_i, the beat instants chosen after t_i will not influence the choice (or score contribution) of beat times prior to t_i, so the entire best-scoring sequence up to time t_i can be calculated and fixed at time t_i without having to consider any future events
- By contrast, a cost function where events subsequent to t_i could influence the cost contribution of earlier events would not be amenable to this optimization.

Algorithm availability and performance

librosa[14]'s beat_track algorithm[15] is Ellis' Dynamic Programming

MIREX 2006 Audio Beat Tracking Summary Results

Contestant	P-Score (average)
dixon	0.575
davies	0.571
klapuri	0.564
ellis	0.552
brossier	0.453

download these results as csv

MIREX 2006 Audio Beat Tracking Runtime Data

Contestant	Machine	Run-time(seconds)
brossier	LINUX	139
davies	FAST	1394
dixon	FAST	639
ellis	LINUX	498
klapuri	LINUX	1218

download these results as csv

Figure: Results from MIREX 2006[16]

14. Brian McFee et al. 2015. "librosa: Audio and Music Signal Analysis in Python," 18–24. January. https://doi.org/10.25080/Majora-7b98e3ed-003.
15. *librosa.beat.beat_track – librosa 0.8.0 documentation.* https://librosa.org/doc/main/generated/librosa.beat.beat_track.html.

2021-02-22

Dynamic Programming for Beat Tracking

└Algorithm availability and performance

Algorithm availability and performance
librosa[14]'s beat_track algorithm[15] is Ellis' Dynamic Programming
MIREX 2006 Audio Beat Tracking Summary Results
Contestant P-Score (average) dixon 0.575 davies 0.571 klapuri 0.564 ellis 0.552 brossier 0.453
download these results as csv
MIREX 2006 Audio Beat Tracking Runtime Data
Contestant Machine Run-time(seconds) brossier LINUX 139 davies FAST 1394 dixon FAST 639 ellis LINUX 498 klapuri LINUX 1218
download these results as csv
Figure: Results from MIREX 2006[16]
14. Brian McFee et al. 2015. "librosa: Audio and Music Signal Analysis in Python," 18–24. January. https://doi.org/10.25080/Majora-7b98e3ed-003.
15. *librosa.beat.beat_track – librosa 0.8.0 documentation.* https://librosa.org/doc/main/generated/librosa.beat.beat_track.html.
16. *2006:Audio Beat Tracking.* https://www.music-ir.org/mirex/wiki/2006:Audio_

Origins

Invented by Richard Bellman[17]

- I spent the Fall quarter (of 1950) at the RAND Corporation (R&D group employed at the time by the Air Force). My first task was to find a name for multistage decision processes.
- Programming, not like coding, but like scheduling or planning
- Dynamic, multistage, time-varying – and dynamic is impossible to use prejoratively
- Chose the name "dynamic programming" to shield it from the Secretary of Defense, Wilson, who was anti math research:

  > *he actually had a pathological fear and hatred of the word, research. I'm not using the term lightly; I'm using it precisely. His face would suffuse, he would turn red, and he would get violent if people used the term, research, in his presence*

---

17. S.E. Dreyfus. 2002. "Richard Bellman on the Birth of Dynamic Programming." *Operations Research* 50 (February): 48–51. https://doi.org/10.1287/opre.50.1.48.17791. http://www.cas.mcmaster.ca/~se3c03/journal_papers/dy_birth.pdf.

Origins
Invented by Richard Bellman17
I spent the Fall quarter (of 1950) at the RAND Corporation (R&D group employed at the time by the Air Force). My first task was to find a name for multistage decision processes.
Programming, not like coding, but like scheduling or planning
Dynamic, multistage, time-varying – and dynamic is impossible to use prejoratively
Chose the name "dynamic programming" to shield it from the Secretary of Defense, Wilson, who was anti math research:
he actually had a pathological fear and hatred of the word, research. I'm not using the term lightly; I'm using it precisely. His face would suffuse, he would turn red, and he would get violent if people used the term, research, in his presence
17. S.E. Dreyfus. 2002. "Richard Bellman on the Birth of Dynamic Programming." Operations Research 50 (February): 48–51. https://doi.org/10.1287/opre.50.1.48.17791. http://www.cas.mcmaster.ca/~se3c03/journal_papers/dy_birth.pdf.

More dynamic programming in MIR

From MIR survey[18]:

- Cambouropoulos, E., Crochemore, M., Iliopoulos, C. S., Mohamed, M., and Sagot, M. (2005) A Pattern Extraction Algorithm for Abstract Melodic Representations that Allow Partial Overlapping of Intervallic Categories. In, T. Crawford, M. Sandler, editors, Proceedings of the 6th International Conference on Music Information Retrieval (ISMIR 2005), pp. 167–174
- Adams, N. H., Bartsch, M., Shifrin J., and Wakefield, G. (2004) Time series alignment for music information retrieval. In Proceedings of the International Conference on Music Information Retrieval, 303–310
- Cambouropoulos, E. (2006) Musical Parallelism and Melodic Segmentation: A Computational Approach. Music Perception 23(3):249-269

---

18. Ryan Demopoulos and Michael Katchabaw. 2007. "Music Information Retrieval: A Survey of Issues and Approaches" (February). https://www.csd.uwo.ca/~mkatchab/pubs/tr677.pdf.

More dynamic programming in MIR
From MIR survey[18]:
- Cambouropoulos, E., Crochemore, M., Iliopoulos, C. S., Mohamed, M., and Sagot, M. (2005) A Pattern Extraction Algorithm for Abstract Melodic Representations that Allow Partial Overlapping of Intervallic Categories. In, T. Crawford, M. Sandler, editors, Proceedings of the 6th International Conference on Music Information Retrieval (ISMIR 2005), pp. 167–174
- Adams, N. H., Bartsch, M., Shifrin J., and Wakefield, G. (2004) Time series alignment for music information retrieval. In Proceedings of the International Conference on Music Information Retrieval, 303–310
- Cambouropoulos, E. (2006) Musical Parallelism and Melodic Segmentation: A Computational Approach. Music Perception 23(3):249-269
18. Ryan Demopoulos and Michael Katchabaw. 2007. "Music Information Retrieval: A Survey of Issues and Approaches" (February). https://www.csd.uwo.ca/~mkatchab/pubs/tr677.pdf.