Dynamic Programming

Dynamic Programming is mainly an optimization over plain [recursion](https://www.geeksforgeeks.org/recursion/). Wherever we see a recursive solution that has repeated calls for same inputs, we can optimize it using Dynamic Programming. The idea is to simply store the results of subproblems, so that we do not have to re-comupute them when needed later. This simple optimization reduces time complexities from exponential to polynomial. For example, if we write simple recursive solution for [Fibonacci Numbers](https://www.geeksforgeeks.org/program-for-nth-fibonacci-number/), we get exponential time complexity and if we optimize it by storing solutions of subproblems, time complexity reduces to linear.

In terms of mathematical optimization, dynamic programming usually refers to simplifying a decision by breaking it down into a sequence of decision steps over time. This is done by defining a sequence of **value functions** *V*1, *V*2, ..., *Vn* taking *y* as an argument representing the [**state**](https://en.wikipedia.org/wiki/State_variable) of the system at times *i* from 1 to *n*. The definition of *Vn*(*y*) is the value obtained in state *y* at the last time *n*. The values *Vi* at earlier times *i* = *n* −1, *n* − 2, ..., 2, 1 can be found by working backwards, using a [recursive](https://en.wikipedia.org/wiki/Recursion) relationship called the [Bellman equation](https://en.wikipedia.org/wiki/Bellman_equation). For *i* = 2, ..., *n*, *Vi*−1 at any state *y* is calculated from *Vi* by maximizing a simple function (usually the sum) of the gain from a decision at time *i* − 1 and the function *Vi* at the new state of the system if this decision is made. Since *Vi* has already been calculated for the needed states, the above operation yields *Vi*−1 for those states. Finally, *V*1 at the initial state of the system is the value of the optimal solution. The optimal values of the decision variables can be recovered, one by one, by tracking back the calculations already performed.

### Control theory**[**[**edit**](https://en.wikipedia.org/w/index.php?title=Dynamic_programming&action=edit&section=3)**]**

In [control theory](https://en.wikipedia.org/wiki/Control_theory), a typical problem is to find an admissible control {\displaystyle \mathbf {u} ^{\ast }} which causes the system {\displaystyle {\dot {\mathbf {x} }}(t)=\mathbf {g} \left(\mathbf {x} (t),\mathbf {u} (t),t\right)} to follow an admissible trajectory {\displaystyle \mathbf {x} ^{\ast }} on a continuous time interval {\displaystyle t\_{0}\leq t\leq t\_{1}} that minimizes a [cost function](https://en.wikipedia.org/wiki/Loss_function)

{\displaystyle J=b\left(\mathbf {x} (t\_{1}),t\_{1}\right)+\int \_{t\_{0}}^{t\_{1}}f\left(\mathbf {x} (t),\mathbf {u} (t),t\right)\mathrm {d} t}

The solution to this problem is an optimal control law or policy {\displaystyle \mathbf {u} ^{\ast }=h(\mathbf {x} (t),t)}, which produces an optimal trajectory {\displaystyle \mathbf {x} ^{\ast }} and an optimized loss function {\displaystyle J^{\ast }}. The latter obeys the fundamental equation of dynamic programming:

{\displaystyle -J\_{t}^{\ast }=\min \_{\mathbf {u} }\left\{f\left(\mathbf {x} (t),\mathbf {u} (t),t\right)+J\_{x}^{\ast {\mathsf {T}}}\mathbf {g} \left(\mathbf {x} (t),\mathbf {u} (t),t\right)\right\}}

a [partial differential equation](https://en.wikipedia.org/wiki/Partial_differential_equation) known as the [Hamilton–Jacobi–Bellman equation](https://en.wikipedia.org/wiki/Hamilton%E2%80%93Jacobi%E2%80%93Bellman_equation), in which {\displaystyle J\_{x}^{\ast }={\frac {\partial J^{\ast }}{\partial \mathbf {x} }}=\left[{\frac {\partial J^{\ast }}{\partial x\_{1}}}~~~~{\frac {\partial J^{\ast }}{\partial x\_{2}}}~~~~\dots ~~~~{\frac {\partial J^{\ast }}{\partial x\_{n}}}\right]^{\mathsf {T}}} and {\displaystyle J\_{t}^{\ast }={\frac {\partial J^{\ast }}{\partial t}}}. One finds the minimizing {\displaystyle \mathbf {u} } in terms of {\displaystyle t}, {\displaystyle \mathbf {x} }, and the unknown function {\displaystyle J\_{x}^{\ast }} and then substitutes the result into the Hamilton–Jacobi–Bellman equation to get the partial differential equation to be solved with boundary condition {\displaystyle J\left(t\_{1}\right)=b\left(\mathbf {x} (t\_{1}),t\_{1}\right)}.[[2]](https://en.wikipedia.org/wiki/Dynamic_programming#cite_note-2) In practice, this generally requires [numerical techniques](https://en.wikipedia.org/wiki/Numerical_partial_differential_equations) for some discrete approximation to the exact optimization relationship.

Alternatively, the continuous process can be approximated by a discrete system, which leads to a following recurrence relation analog to the Hamilton–Jacobi–Bellman equation:

{\displaystyle J\_{k}^{\ast }\left(\mathbf {x} \_{n-k}\right)=\min \_{\mathbf {u} \_{n-k}}\left\{{\hat {f}}\left(\mathbf {x} \_{n-k},\mathbf {u} \_{n-k}\right)+J\_{k-1}^{\ast }\left({\hat {g}}\left(\mathbf {x} \_{n-k},\mathbf {u} \_{n-k}\right)\right)\right\}}

at the {\displaystyle k}-th stage of {\displaystyle n} equally spaced discrete time intervals, and where {\displaystyle {\hat {f}}} and {\displaystyle {\hat {g}}} denote discrete approximations to {\displaystyle f} and {\displaystyle \mathbf {g} }. This functional equation is known as the [Bellman equation](https://en.wikipedia.org/wiki/Bellman_equation), which can be solved for an exact solution of the discrete approximation of the optimization equation.[[3]](https://en.wikipedia.org/wiki/Dynamic_programming#cite_note-3)

<https://medium.com/basecs/speeding-up-the-traveling-salesman-using-dynamic-programming-b76d7552e8dd>

for data set

|  |
| --- |
| 0 107 241 190 124 80 316 76 152 157 283 133 113 297 228 129 348 276 188 150 65 341 184 67 221 169 108 45 167 |
|  | 107 0 148 137 88 127 336 183 134 95 254 180 101 234 175 176 265 199 182 67 42 278 271 146 251 105 191 139 79 |
|  | 241 148 0 374 171 259 509 317 217 232 491 312 280 391 412 349 422 356 355 204 182 435 417 292 424 116 337 273 77 |
|  | 190 137 374 0 202 234 222 192 248 42 117 287 79 107 38 121 152 86 68 70 137 151 239 135 137 242 165 228 205 |
|  | 124 88 171 202 0 61 392 202 46 160 319 112 163 322 240 232 314 287 238 155 65 366 300 175 307 57 220 121 97 |
|  | 80 127 259 234 61 0 386 141 72 167 351 55 157 331 272 226 362 296 232 164 85 375 249 147 301 118 188 60 185 |
|  | 316 336 509 222 392 386 0 233 438 254 202 439 235 254 210 187 313 266 154 282 321 298 168 249 95 437 190 314 435 |
|  | 76 183 317 192 202 141 233 0 213 188 272 193 131 302 233 98 344 289 177 216 141 346 108 57 190 245 43 81 243 |
|  | 152 134 217 248 46 72 438 213 0 206 365 89 209 368 286 278 360 333 284 201 111 412 321 221 353 72 266 132 111 |
|  | 157 95 232 42 160 167 254 188 206 0 159 220 57 149 80 132 193 127 100 28 95 193 241 131 169 200 161 189 163 |
|  | 283 254 491 117 319 351 202 272 365 159 0 404 176 106 79 161 165 141 95 187 254 103 279 215 117 359 216 308 322 |
|  | 133 180 312 287 112 55 439 193 89 220 404 0 210 384 325 279 415 349 285 217 138 428 310 200 354 169 241 112 238 |
|  | 113 101 280 79 163 157 235 131 209 57 176 210 0 186 117 75 231 165 81 85 92 230 184 74 150 208 104 158 206 |
|  | 297 234 391 107 322 331 254 302 368 149 106 384 186 0 69 191 59 35 125 167 255 44 309 245 169 327 246 335 288 |
|  | 228 175 412 38 240 272 210 233 286 80 79 325 117 69 0 122 122 56 56 108 175 113 240 176 125 280 177 266 243 |
|  | 129 176 349 121 232 226 187 98 278 132 161 279 75 191 122 0 244 178 66 160 161 235 118 62 92 277 55 155 275 |
|  | 348 265 422 152 314 362 313 344 360 193 165 415 231 59 122 244 0 66 178 198 286 77 362 287 228 358 299 380 319 |
|  | 276 199 356 86 287 296 266 289 333 127 141 349 165 35 56 178 66 0 112 132 220 79 296 232 181 292 233 314 253 |
|  | 188 182 355 68 238 232 154 177 284 100 95 285 81 125 56 66 178 112 0 128 167 169 179 120 69 283 121 213 281 |
|  | 150 67 204 70 155 164 282 216 201 28 187 217 85 167 108 160 198 132 128 0 88 211 269 159 197 172 189 182 135 |
|  | 65 42 182 137 65 85 321 141 111 95 254 138 92 255 175 161 286 220 167 88 0 299 229 104 236 110 149 97 108 |
|  | 341 278 435 151 366 375 298 346 412 193 103 428 230 44 113 235 77 79 169 211 299 0 353 289 213 371 290 379 332 |
|  | 184 271 417 239 300 249 168 108 321 241 279 310 184 309 240 118 362 296 179 269 229 353 0 121 162 345 80 189 342 |
|  | 67 146 292 135 175 147 249 57 221 131 215 200 74 245 176 62 287 232 120 159 104 289 121 0 154 220 41 93 218 |
|  | 221 251 424 137 307 301 95 190 353 169 117 354 150 169 125 92 228 181 69 197 236 213 162 154 0 352 147 247 350 |
|  | 169 105 116 242 57 118 437 245 72 200 359 169 208 327 280 277 358 292 283 172 110 371 345 220 352 0 265 178 39 |
|  | 108 191 337 165 220 188 190 43 266 161 216 241 104 246 177 55 299 233 121 189 149 290 80 41 147 265 0 124 263 |
|  | 45 139 273 228 121 60 314 81 132 189 308 112 158 335 266 155 380 314 213 182 97 379 189 93 247 178 124 0 199 |
|  | 167 79 77 205 97 185 435 243 111 163 322 238 206 288 243 275 319 253 281 135 108 332 342 218 350 39 263 199 0 |

////////////////////////////

Enter the number of villages: 29

Enter the Cost Matrix

Enter Elements of Row: 1

0 107 241 190 124 80 316 76 152 157 283 133 113 297 228 129 348 276 188 150 65 341 184 67 221 169 108 45 167

Enter Elements of Row: 2

107 0 148 137 88 127 336 183 134 95 254 180 101 234 175 176 265 199 182 67 42 278 271 146 251 105 191 139 79

Enter Elements of Row: 3

241 148 0 374 171 259 509 317 217 232 491 312 280 391 412 349 422 356 355 204 182 435 417 292 424 116 337 273 77

Enter Elements of Row: 4

190 137 374 0 202 234 222 192 248 42 117 287 79 107 38 121 152 86 68 70 137 151 239 135 137 242 165 228 205

Enter Elements of Row: 5

124 88 171 202 0 61 392 202 46 160 319 112 163 322 240 232 314 287 238 155 65 366 300 175 307 57 220 121 97

Enter Elements of Row: 6

80 127 259 234 61 0 386 141 72 167 351 55 157 331 272 226 362 296 232 164 85 375 249 147 301 118 188 60 185

Enter Elements of Row: 7

316 336 509 222 392 386 0 233 438 254 202 439 235 254 210 187 313 266 154 282 321 298 168 249 95 437 190 314 435

Enter Elements of Row: 8

76 183 317 192 202 141 233 0 213 188 272 193 131 302 233 98 344 289 177 216 141 346 108 57 190 245 43 81 243

Enter Elements of Row: 9

152 134 217 248 46 72 438 213 0 206 365 89 209 368 286 278 360 333 284 201 111 412 321 221 353 72 266 132 111

Enter Elements of Row: 10

157 95 232 42 160 167 254 188 206 0 159 220 57 149 80 132 193 127 100 28 95 193 241 131 169 200 161 189 163

Enter Elements of Row: 11

283 254 491 117 319 351 202 272 365 159 0 404 176 106 79 161 165 141 95 187 254 103 279 215 117 359 216 308 322

Enter Elements of Row: 12

133 180 312 287 112 55 439 193 89 220 404 0 210 384 325 279 415 349 285 217 138 428 310 200 354 169 241 112 238

Enter Elements of Row: 13

113 101 280 79 163 157 235 131 209 57 176 210 0 186 117 75 231 165 81 85 92 230 184 74 150 208 104 158 206

Enter Elements of Row: 14

297 234 391 107 322 331 254 302 368 149 106 384 186 0 69 191 59 35 125 167 255 44 309 245 169 327 246 335 288

Enter Elements of Row: 15

228 175 412 38 240 272 210 233 286 80 79 325 117 69 0 122 122 56 56 108 175 113 240 176 125 280 177 266 243

Enter Elements of Row: 16

129 176 349 121 232 226 187 98 278 132 161 279 75 191 122 0 244 178 66 160 161 235 118 62 92 277 55 155 275

Enter Elements of Row: 17

348 265 422 152 314 362 313 344 360 193 165 415 231 59 122 244 0 66 178 198 286 77 362 287 228 358 299 380 319

Enter Elements of Row: 18

276 199 356 86 287 296 266 289 333 127 141 349 165 35 56 178 66 0 112 132 220 79 296 232 181 292 233 314 253

Enter Elements of Row: 19

188 182 355 68 238 232 154 177 284 100 95 285 81 125 56 66 178 112 0 128 167 169 179 120 69 283 121 213 281

Enter Elements of Row: 20

150 67 204 70 155 164 282 216 201 28 187 217 85 167 108 160 198 132 128 0 88 211 269 159 197 172 189 182 135

Enter Elements of Row: 21

65 42 182 137 65 85 321 141 111 95 254 138 92 255 175 161 286 220 167 88 0 299 229 104 236 110 149 97 108

Enter Elements of Row: 22

341 278 435 151 366 375 298 346 412 193 103 428 230 44 113 235 77 79 169 211 299 0 353 289 213 371 290 379 332

Enter Elements of Row: 23

184 271 417 239 300 249 168 108 321 241 279 310 184 309 240 118 362 296 179 269 229 353 0 121 162 345 80 189 342

Enter Elements of Row: 24

67 146 292 135 175 147 249 57 221 131 215 200 74 245 176 62 287 232 120 159 104 289 121 0 154 220 41 93 218

Enter Elements of Row: 25

221 251 424 137 307 301 95 190 353 169 117 354 150 169 125 92 228 181 69 197 236 213 162 154 0 352 147 247 350

Enter Elements of Row: 26

169 105 116 242 57 118 437 245 72 200 359 169 208 327 280 277 358 292 283 172 110 371 345 220 352 0 265 178 39

Enter Elements of Row: 27

108 191 337 165 220 188 190 43 266 161 216 241 104 246 177 55 299 233 121 189 149 290 80 41 147 265 0 124 263

Enter Elements of Row: 28

45 139 273 228 121 60 314 81 132 189 308 112 158 335 266 155 380 314 213 182 97 379 189 93 247 178 124 0 199

Enter Elements of Row: 29

167 79 77 205 97 185 435 243 111 163 322 238 206 288 243 275 319 253 281 135 108 332 342 218 350 39 263 199 0

The cost list is:

0 0 0 0 0 0 0 0 0 0 0 0 0 137

88 127 336 183 134 95 241 148 0 374 171 259 509 317 217

0 0 0 137 88 127 336 183 134 95 241 148 0 374

171 259 509 317 217 232 190 137 374 0 202 234 222 192 248

241 148 0 374 171 259 509 317 217 232 190 137 374 0

202 234 222 192 248 42 124 88 171 202 0 61 392 202 46

190 137 374 0 202 234 222 192 248 42 124 88 171 202

0 61 392 202 46 160 80 127 259 234 61 0 386 141 72

124 88 171 202 0 61 392 202 46 160 80 127 259 234

61 0 386 141 72 167 316 336 509 222 392 386 0 233 438

80 127 259 234 61 0 386 141 72 167 316 336 509 222

392 386 0 233 438 254 76 183 317 192 202 141 233 0 213

316 336 509 222 392 386 0 233 438 254 76 183 317 192

202 141 233 0 213 188 152 134 217 248 46 72 438 213 0

76 183 317 192 202 141 233 0 213 188 152 134 217 248

46 72 438 213 0 206 157 95 232 42 160 167 254 188 206

152 134 217 248 46 72 438 213 0 206 157 95 232 42

160 167 254 188 206 0 283 254 491 117 319 351 202 272 365

157 95 232 42 160 167 254 188 206 0 283 254 491 117

319 351 202 272 365 159 133 180 312 287 112 55 439 193 89

283 254 491 117 319 351 202 272 365 159 133 180 312 287

112 55 439 193 89 220 113 101 280 79 163 157 235 131 209

133 180 312 287 112 55 439 193 89 220 113 101 280 79

163 157 235 131 209 57 297 234 391 107 322 331 254 302 368

113 101 280 79 163 157 235 131 209 57 297 234 391 107

322 331 254 302 368 149 228 175 412 38 240 272 210 233 286

297 234 391 107 322 331 254 302 368 149 228 175 412 38

240 272 210 233 286 80 129 176 349 121 232 226 187 98 278

228 175 412 38 240 272 210 233 286 80 129 176 349 121

232 226 187 98 278 132 348 265 422 152 314 362 313 344 360

129 176 349 121 232 226 187 98 278 132 348 265 422 152

314 362 313 344 360 193 276 199 356 86 287 296 266 289 333

348 265 422 152 314 362 313 344 360 193 276 199 356 86

287 296 266 289 333 127 188 182 355 68 238 232 154 177 284

276 199 356 86 287 296 266 289 333 127 188 182 355 68

238 232 154 177 284 100 150 67 204 70 155 164 282 216 201

188 182 355 68 238 232 154 177 284 100 150 67 204 70

155 164 282 216 201 28 65 42 182 137 65 85 321 141 111

150 67 204 70 155 164 282 216 201 28 65 42 182 137

65 85 321 141 111 95 341 278 435 151 366 375 298 346 412

65 42 182 137 65 85 321 141 111 95 341 278 435 151

366 375 298 346 412 193 184 271 417 239 300 249 168 108 321

341 278 435 151 366 375 298 346 412 193 184 271 417 239

300 249 168 108 321 241 67 146 292 135 175 147 249 57 221

184 271 417 239 300 249 168 108 321 241 67 146 292 135

175 147 249 57 221 131 221 251 424 137 307 301 95 190 353

67 146 292 135 175 147 249 57 221 131 221 251 424 137

307 301 95 190 353 169 169 105 116 242 57 118 437 245 72

221 251 424 137 307 301 95 190 353 169 169 105 116 242

57 118 437 245 72 200 108 191 337 165 220 188 190 43 266

169 105 116 242 57 118 437 245 72 200 108 191 337 165

220 188 190 43 266 161 45 139 273 228 121 60 314 81 132

108 191 337 165 220 188 190 43 266 161 45 139 273 228

121 60 314 81 132 189 167 79 77 205 97 185 435 243 111

45 139 273 228 121 60 314 81 132 189 167 79 77 205

97 185 435 243 111 163 322 238 206 288 243 275 319 253 281

167 79 77 205 97 185 435 243 111 163 322 238 206 288

243 275 319 253 281 135 108 332 342 218 350 39 263 199 0

The Path is:

1--->20--->2--->21--->6--->9--->5--->12--->24--->8--->16--->4--->29--->26--->25--->19--->28--->13--->14--->27--->23--->18--->22--->17--->10--->15--->11--->7--->1

Minimum cost is 3383

Process returned 0 (0x0) execution time : 3.184 s

Press any key to continue.