incremental A*

a path from start to end has been found by A*

but now the problem changes

in the example of trucks and parcels: a truck is out of order, a route is closed, a warehou:

optimal plans may not still be optimal
may not be executable at all!

---

incremental A*: settings

A* has finished

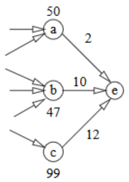
but now:

1. some $n \rightarrow m$ become unavailable
2. some new $n \rightarrow m$ are created
3. more generally, the cost of some $n \rightarrow m$ change

search from scratch
or use data obtained by the previous A* search

[italian] from scratch = dall'inizio, senza sfruttare quello che si era fatto in precedenza

obvious case



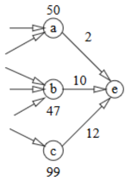
situation at end of A*

meaning of numbers: `d(a)=50`, `a→e=2`, etc.

optimal path?

---

optimal path



best path: go to `a`, then `e`
cost is `d(a)+a→e = 50+2`

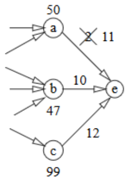
cost of the alternatives: `47+10=57` and `99+12=111`
worse

how to get to `a`?
same method: choose cheapest predecessor + step

what if `a→e` increases?

---

increased cost at the end



cost of `a→e` increases

alternative paths to `e`:

- `d(a) + a→e = 50+11 = 61`
- `d(b) + b→e = 47+10 = 57`
- `d(c) + c→e = 99+12 = 111`

go to `b`, then to `e`

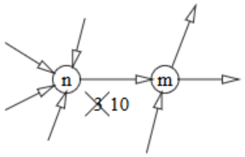
inner nodes?

data from previous search may be useful
proved when changing `a→e`

`e` is the target state
what if an inner step changes cost?

---

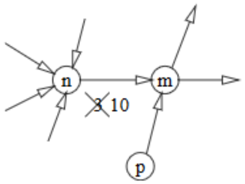
arbitrary change of cost



example: `n→m` changes from `3` to `10`
both `n` and `m` are linked to other states!

---

effect of change



previously: `n→m` may the optimal way to `m`

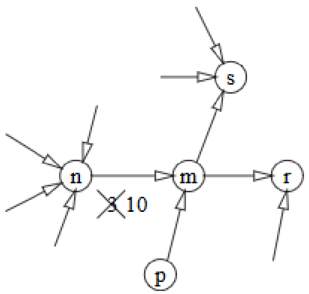
now: `p→m` may be

happens if: `d(p) + p→m < d(n) + n→m`

means: `start⇒p→m` cheaper than `start⇒n→m`

not the only change!

chain reaction



`d(m)` = best known `start⇒m`
previously `d(m)=d(n)+n→m` (with `n→m=3`)
now `d(m)=d(p)+p→m` (greater)

affects the successors of `m`
maybe `start⇒m→r` no longer the cheapest way to `s`

---

how to propagate the changes

seen before:
`d(_)` correct ⇒ minimal path

`n→m` changed, `d(_)` invalid
fix it

[note] As seen before, if `d(_)` is valid an optimal path can be found by starting from the end and then repeatedly going back from the current node `m` to the precessor `n` with a minimal `d(n) + n→m`.

When some costs `n→m` change, `d(_)` may not longer be correct. The previously optimal plan may no longer be optimal.

fix invalid distances

this is what A* does

invariant in A*:

> `d(m)` is the minimal cost of a path of closed states from the start to `m`

“path of closed states” = all states closed but possibly `m`

at each step, A* maintain this condition true
while attempting to enlarge the set of closed nodes

it “fixes” `d(_)`

[note] The invariant is valid for every state `m`, including the ones that have not not been reached at all. This is because the algorithm initializes `d(m)=∞`, which is the correct distance for a state that is not reached by a path of closed states.

---

the invariant in A*

> `d(m)` is the minimal cost of a path of closed states from the start to `m`

when `n` enters the set of closed nodes,
if `start⇒n` is a path of closed nodes and `n→m` exists:

- since the invariant was true before and `start⇒n` is a path of closed nodes then `d(n)` is correct
- path `start⇒n→m` is now a path of closed nodes to `m`
- its cost is `d(n) + n→m` since `d(n)` is correct
- if the current value of `d(m)` is less than or equal `d(n) + n→m`,
  this value is now wrong
  (otherwise, the following steps are not done)
- set `d(m) = d(n) + n→m`
- if `m` is closed, now `start⇒n→m→r` is a new path of closed nodes to every successor `r` of `m`
- as a result, `d(r)` may now be wrong
- to fix the problem, remove `m` from the closed states

A* tries to enlarge the set of closed nodes
ensuring that the invariant is always correct

fixing distances by A*

no need for a special algorithm for fixing distances
A* itself makes distances valid

works without changes if `n→m` may only decrease

just reopen `m` and continue

---

increasing costs

rule for always decreasing distances:

> if `d(n)+n→m < d(m)`, update `d(m)` and reopen `m`

what it does:
ensure that `d(_)` is the minimal cost of a path of closed states

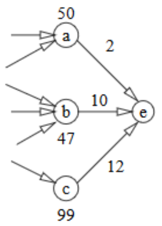
new path found ⇒ maybe better than the previous

in general?

---

minimal path

seen before:



if `a→e`, `b→e`, `c→e` exist
minimal cost of path to `e` is:

```
min(d(a)+a→e, d(b)+b→e, d(c)+c→e)
```

not only if `e` is the target
true in general

why: `d(a)` is the minimal cost for reaching `a`, etc.

incremental A*

instead of: if `d(m) > d(n)+n→m` …

use the new rule:

- `cp(m) = { n | n→m, n ∈ closed }`
- `temp = min`$_{n \in cp(m)}$ `d(n)+n→m`
- if `temp ≠ d(m)` then:
  - set `d(m) = temp`
  - remove `m` from `closed`

---

reopening

A* tries to reach the target by closed states

by the invariant `d(end)` would be correct
allows going back by the cheapest path

removing states from the set of closed is in the opposite direction
BUT: necessary to maintain the invariant

---

actual algorithms

LPA*, D*, …