Algoritmo di Programmazione Dinamica per KP01

- Dati 2 interi m, $\hat{c}$:

$$1 \leq m \leq n, \qquad 0 \leq \hat{c} \leq c$$

consideriamo il sottoproblema:

$$f_m(\hat{c}) = \max \left\{ \sum_{j=1}^{m} p_j x_j \;:\; \sum_{j=1}^{m} w_j x_j \leq \hat{c}, x_j \in \{0,1\}, j = 1, \ldots, m \right\}$$

- Chiaramente:

$$f_1(\hat{c}) = \begin{cases} 0 & \text{for } \hat{c} = 0, \ldots, w_1 - 1 \\ p_1 & \text{for } \hat{c} = w_1, \ldots, c \end{cases}$$

Per i valori successivi:

$$f_m(\hat{c}) = \begin{cases} f_{m-1}(\hat{c}) & \text{for } \hat{c} = 0, \ldots, w_m - 1 \\ \max(f_{m-1}(\hat{c}),\ f_{m-1}(\hat{c} - w_m) + p_m) & \text{for } \hat{c} = w_m, \ldots, c \end{cases}$$

- **procedure KP01_DP**:

begin
  for $\hat{c} := 0$ **to** $w_1 - 1$ **do** $f_1(\hat{c}) := 0$;
  for $\hat{c} := w_1$ **to** c **do** $f_1(\hat{c}) := p_1$;
  for $m := 2$ **to** n **do**
    for $\hat{c} := 0$ **to** $w_m - 1$ **do** $f_m(\hat{c}) := f_{m-1}(\hat{c})$;
    for $\hat{c} := w_m$ **to** c **do**
      if $f_{m-1}(\hat{c} - w_m) + p_m > f_{m-1}(\hat{c})$ **then**
        $f_m(\hat{c}) := f_{m-1}(\hat{c} - w_m) + p_m$
      else $f_m(\hat{c}) := f_{m-1}(\hat{c})$
end.

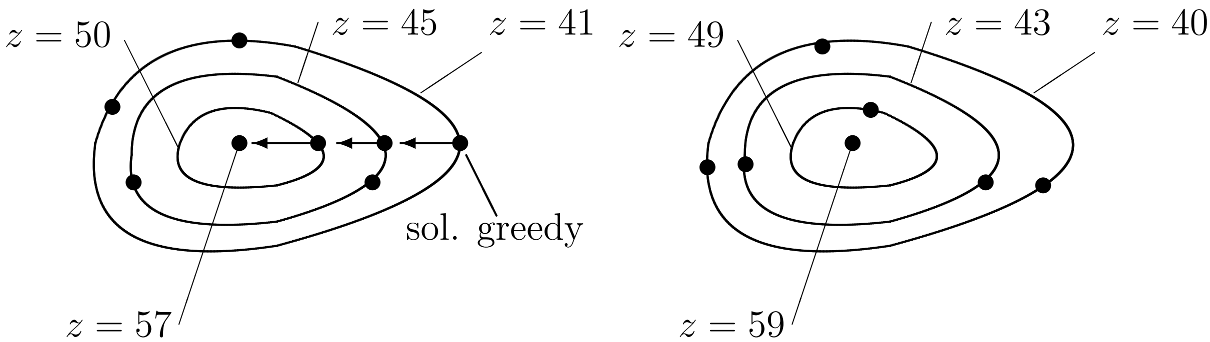
Diversi tipi di algoritmi approssimati

- *Algoritmi greedy*: trovano una soluzione mediante semplice scansione dei dati (algoritmi veloci e poco accurati).
- *Algoritmi "local search"*: partono da una soluzione iniziale e ne esplorano un *intorno*, ossia generano una serie di soluzioni ottenute una dall'altra attraverso "piccoli" miglioramenti, terminando quando non sono più possibili miglioramenti. Es:

procedure LOCAL (algoritmo local search per *KP01*)
begin
  esegui GREEDY (z = val. soluzione, $\overline{c}$ = capacità residua);
  while $\emptyset \neq J = \{(i,j) : x_i = 1, x_j = 0, \overline{c} + w_i \geq w_j, p_j > p_i\}$ **do**
    sia $(i^*,\ j^*)$ la coppia : $p_{j^*} - p_{i^*} = \max_{(i,j)\in J}\{p_j - p_i\}$;
    $x_{i^*} := 0, x_{j^*} := 1, z := z - p_{i^*} + p_{j^*}, \overline{c} := \overline{c} + w_{i^*} - w_{j^*}$ (#)
  endwhile
end.

(#) Il cambiamento di soluzione viene detto *mossa*.

L'algoritmo può restare intrappolato in un minimo locale. Es:



Spazio delle soluzioni ammissibili e curve isocosto

- *Algoritmi metaeuristici* per ovviare a questo inconveniente.

Algoritmi metaeuristici

- Principale tecnica metaeuristica: *Tabu Search*.

- Strategia generale: si esegue sempre la miglior mossa trovata, <u>anche se</u> peggiora la soluzione attuale (*uphill move*).
  (Es. LOCAL: $J = \{(i, j) : x_i = 1, x_j = 0, \overline{c} + w_i \geq w_j\}$)

- In pratica, si alterna iterativamente tra:
  – local search per la ricerca di un ottimo locale e, trovatolo,
  – esecuzione della miglior mossa possibile (peggiorativa).
  (Occorre mantenere in memoria la miglior soluzione trovata.)

- In questo modo però la miglior mossa dal miglior vicino dell'ottimo locale riporterebbe all'ottimo locale appena abbandonato. $\Rightarrow$

- *Tabu list:* conserva informazioni sulle mosse più recenti; sono <u>proibite</u> mosse che riportino a soluzioni visitate di recente.

- Principali altri ingredienti di un algoritmo Tabu Search:

– *Aspiration criteria*: condizioni in cui si può violare il tabu;

– *Diversification*: se si ritiene che non ci siano buoni ottimi locali "in zona", si cambia drasticamente la soluzione corrente;

– *Intensification*: si costringe la ricerca locale a rimanere in prossimità di una buona soluzione trovata precedentemente.

- Altre tecniche metaeuristiche:
  – *Algoritmi Randomizzati*: valori casuali per differenziare;
  – *Simulated Annealing*: può accettare mosse peggioranti in qualunque momento ($\leftarrow$ Termodinamica);
  – *Algoritmi Genetici*: basati su analogie con l'evoluzione;
  ...