Algoritmi Euristici

Corso di Laurea in Informatica e Corso di Laurea in Matematica

Roberto Cordone

DI - Università degli Studi di Milano

Lezioni:	Lunedì 13.30 - 15.30 in Aula G30
	Giovedì 13.30 - 15.30 in Aula G30
Ricevimento:	su appuntamento
Tel.:	02 503 16235
E-mail:	roberto.cordone@unimi.it
Web page:	http://homes.di.unimi.it/~cordone/courses/2018-ae/2018-ae.html

Very Large Scale Neighbourhood Search

L'euristica *steepest descent* è

- molto efficace se i bacini di attrazione sono pochi e ampi
- poco efficace se i bacini di attrazione sono molti e piccoli

Allargando l'intorno, in genere si ottengono bacini di attrazione più ampi, ma anche tempi di esplorazione maggiori

Gli approcci *Very Large Scale Neighbourhood* (*VLSN*) *Search*

- intorni esponenziali in $|B|$
- visitati in tempo polinomiale

Due strategie evitano che il tempo di calcolo diventi esponenziale

1. si esplora l'intorno in modo euristico, restituendo una soluzione candidata anziché la migliore dell'intorno stesso
2. si sceglie un intorno nel quale l'obiettivo possa essere ottimizzato in tempo polinomiale anche se il numero di soluzioni è esponenziale

Variable Depth Search (*VDS*)

Gli intorni basati su operazioni sono facilmente parametrizzabili

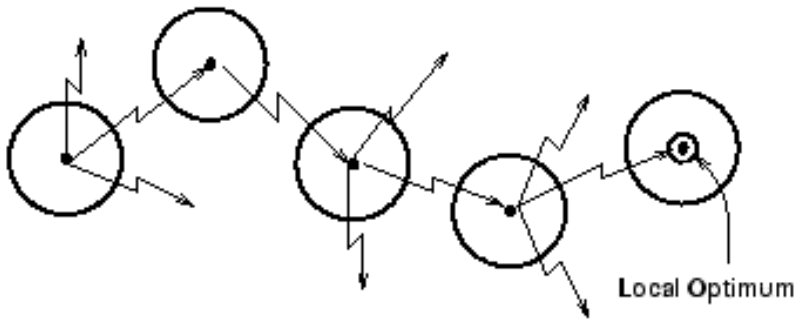
$$N_{\mathcal{O}_k}(x) = \{x' \in X : x' = o_k(o_{k-1}(\ldots o_1(x))) \text{ con } o_1, \ldots, o_k \in \mathcal{O}\}$$

dove k è il numero di operazioni compiute per generare le soluzioni

L'idea è

- definire una mossa composta come sequenza di mosse elementari
- costruire la sequenza ottimizzando ogni passo elementare
- accettare la soluzione finale solo se migliora quella iniziale

La lunghezza k assume valori alti solo se conviene: si adatta a ogni passo



Per gli intorni basati sulla distanza, valgono considerazioni simili

Schema della Variable Depth Search

Data $x^{(t)}$, per ogni $x' \in N\left(x^{(t)}\right)$, anziché limitarsi a valutare $f\left(x'\right)$

1. sceglie la miglior soluzione x'' in un intorno $\hat{N}\left(x'\right) \subseteq N\left(x'\right)$
2. se x'' migliora rispetto alla soluzione iniziale x, esegue la mossa e torna al punto 1; altrimenti termina
3. restituisce la miglior soluzione trovata durante l'intero procedimento

For each $x' \in N\left(x^{(t)}\right)$

{ *Steepest descent* }	{ *Variable Depth Search* }
Compute $f\left(x'\right)$	Compute $f\left(y^*\left(x'\right)\right)$ as follows:
	$y^* := x'$; $y := x'$; Fine := *false*;
	While Fine $=$ *false do*
	$\tilde{y} := \arg \min_{y' \in \hat{N}(y)} f\left(y'\right)$;
	If $f\left(\tilde{y}\right) \geq f\left(x^{(t)}\right)$ *then* Fine := *true*; *else* $y := \tilde{y}$;
	If $f\left(\tilde{y}\right) < f\left(y^*\right)$ *then* $y^* := \tilde{y}$;
	EndWhile;

È una specie di roll-out per algoritmi di scambio

Differenze rispetto alla *steepest descent*

In che cosa differisce rispetto all'esplorazione *steepest descent*?

- trova un ottimo locale per ogni soluzione dell'intorno
  come se eseguisse una specie di *look-ahead* a un passo
- ammette peggioramenti lungo la sequenza di mosse elementari
  (mai rispetto alla soluzione iniziale)
- deve evitare che le mosse elementari della sequenza si elidano a vicenda creando un ciclo (ad es., mosse solo su elementi non toccati)
- spesso usa strategie di modulazione fine per migliorare l'efficienza
  - un intorno ridotto $\tilde{N} \subset \hat{N}$ per scorrere le mosse elementari
  - la strategia *first-best* nell'esplorazione di $\tilde{N}$ (passo elementare)
  - la strategia *first-best* nell'esplorazione di N (intorno iniziale)

L'algoritmo di Lin-Kernighan per il *TSP*

L'intorno $N_{\mathcal{R}_k}(x)$ per il *TSP* simmetrico contiene le soluzioni ottenute

- cancellando k archi di x
- aggiungendo altri k archi che ricostruiscono un circuito hamiltoniano
- eventualmente invertendo tratti del circuito (*ininfluente sul costo*)

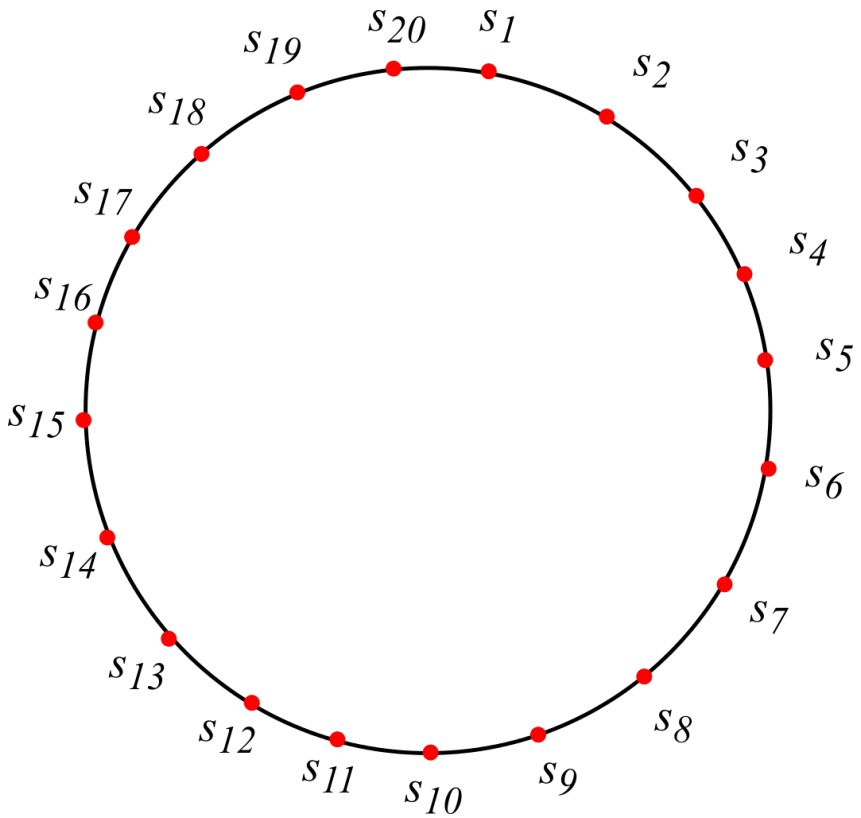
L'algoritmo di Lin-Kernighan applica la *VDS* a sequenze di scambi 2-opt:
ogni scambio k-opt equivale a una sequenza di $(k-1)$ scambi 2-opt, ognuno dei quali cancella uno dei due archi aggiunti dal precedente

Quindi per ogni soluzione $x' \in N_{\mathcal{R}_1}(x)$, ottenuta dallo scambio (i, j)

- si valutano gli scambi 2-opt che cancellano l'arco aggiunto (s_i, s_{j+1}) e ciascun arco di $x \cap x'$ per trovare lo scambio (i', j') migliore
- se migliora rispetto a x, si esegue lo scambio (i', j'), ottenendo x''
- si valutano gli scambi che cancellano l'arco $(s_{i'}, s_{j'+1})$ e ciascun arco di $x \cap x'' \ldots$
- $\ldots$
- se la miglior soluzione fra $x', x'', \ldots$ è meglio di x, la si accetta

Esempio: algoritmo di Lin-Kernighan

Si esplorano tutte le soluzioni $x' \in N_{\mathcal{R}_2}(x)$, ottenute con gli scambi (i, j)

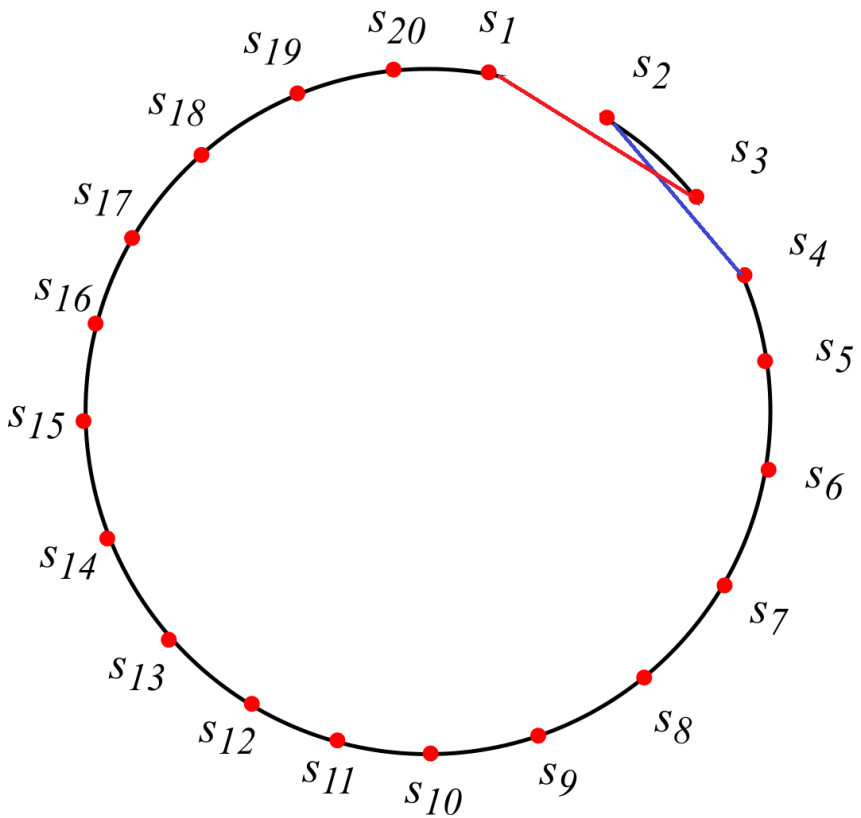


$$x = (s_1\ \overline{s_2\ s_3}\ s_4\ s_5\ s_6\ s_7\ s_8\ s_9\ s_{10}\ s_{11}\ s_{12}\ s_{13}\ s_{14}\ s_{15}\ s_{16}\ s_{17}\ s_{18}\ s_{19}\ s_{20})$$

Concentriamoci sullo scambio $(1, 3)$, che inverte il tratto $(s_2, \ldots, s_3)$

Esempio: algoritmo di Lin-Kernighan

Lo scambio $(1, 3)$ sostituisce (s_1, s_2) e (s_3, s_4) con (s_1, s_3) e (s_2, s_4)

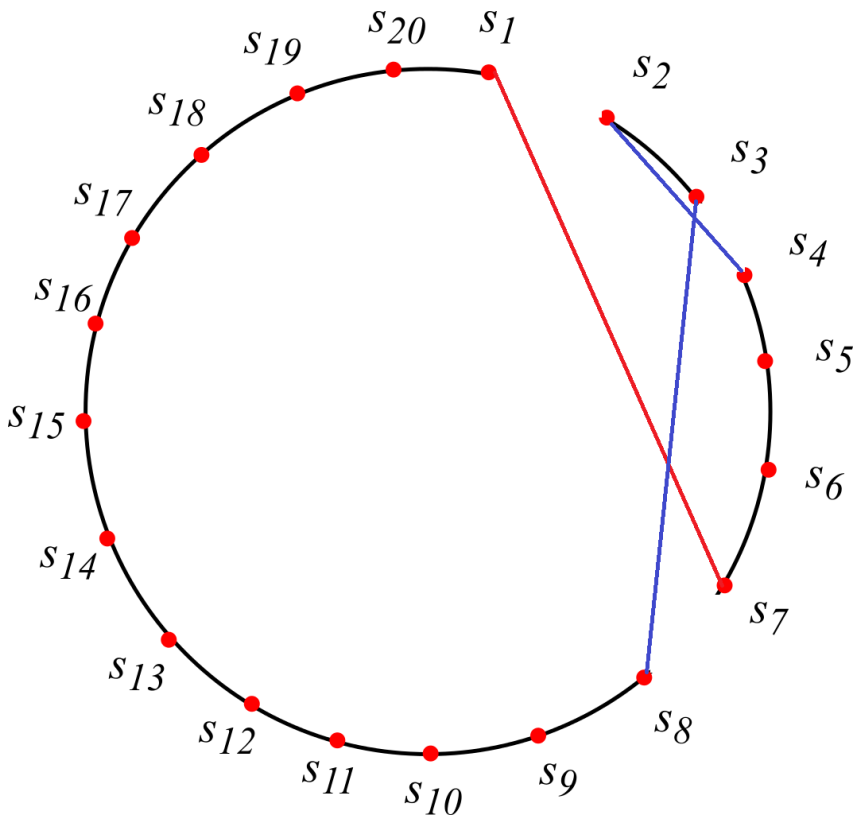


$$x' = (s_1\ \overline{s_3\ s_2\ s_4\ s_5\ s_6\ s_7}\ s_8\ s_9\ s_{10}\ s_{11}\ s_{12}\ s_{13}\ s_{14}\ s_{15}\ s_{16}\ s_{17}\ s_{18}\ s_{19}\ s_{20})$$

Cerchiamo il miglior scambio che rompa (s_1, s_3) e un altro arco di $x \cap x'$
Supponiamo che sia lo scambio $(1, 7)$, che inverte il tratto $(s_3, \ldots, s_7)$

Esempio: algoritmo di Lin-Kernighan

Lo scambio $(1, 7)$ sostituisce (s_1, s_3) e (s_7, s_8) con (s_1, s_7) e (s_3, s_8)

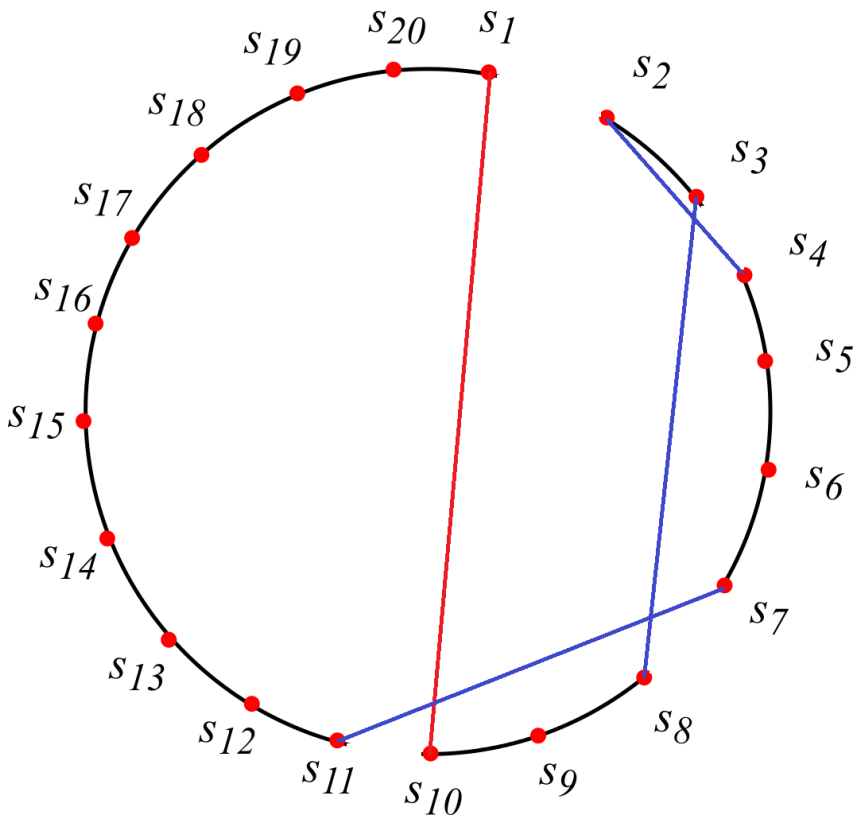


$$x'' = (s_1\ \overline{s_7\ s_6\ s_5\ s_4\ s_2\ s_3\ s_8\ s_9\ s_{10}}\ s_{11}\ s_{12}\ s_{13}\ s_{14}\ s_{15}\ s_{16}\ s_{17}\ s_{18}\ s_{19}\ s_{20})$$

Cerchiamo il miglior scambio che rompa (s_1, s_7) e un altro arco di $x \cap x''$
Supponiamo che sia lo scambio $(1, 10)$, che inverte il tratto $(s_7, \ldots, s_{10})$

Esempio: algoritmo di Lin-Kernighan

Lo scambio $(1, 10)$ sostituisce (s_1, s_7) e (s_{10}, s_{11}) con (s_1, s_{10}) e (s_7, s_{11})

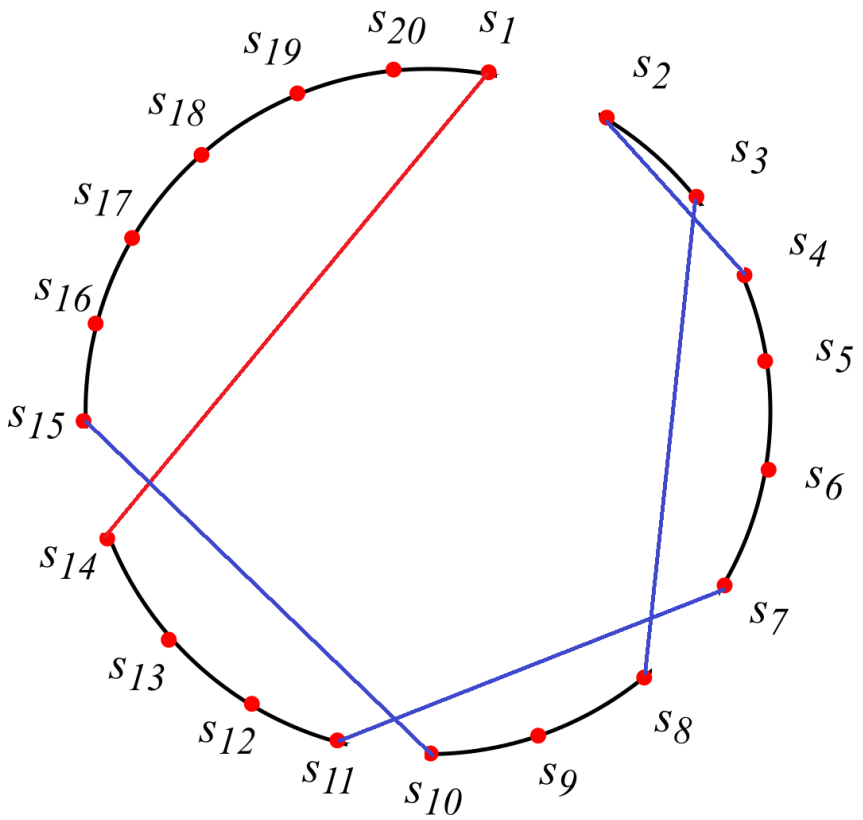


$$x''' = (s_1 \; \overline{s_{10}\, s_9\, s_8\, s_3\, s_2\, s_4\, s_5\, s_6\, s_7\, s_{11}\, s_{12}\, s_{13}\, s_{14}} \; s_{15}\, s_{16}\, s_{17}\, s_{18}\, s_{19}\, s_{20})$$

Cerchiamo il miglior scambio che rompa (s_1, s_{10}) e un altro arco di $x \cap x'''$
Supponiamo che sia lo scambio $(1, 14)$, che inverte il tratto $(s_{10}, \ldots, s_{14})$

Esempio: algoritmo di Lin-Kernighan

Lo scambio $(1, 14)$ sostituisce (s_1, s_{10}) e (s_{14}, s_{15}) con (s_1, s_{14}) e (s_{10}, s_{15})

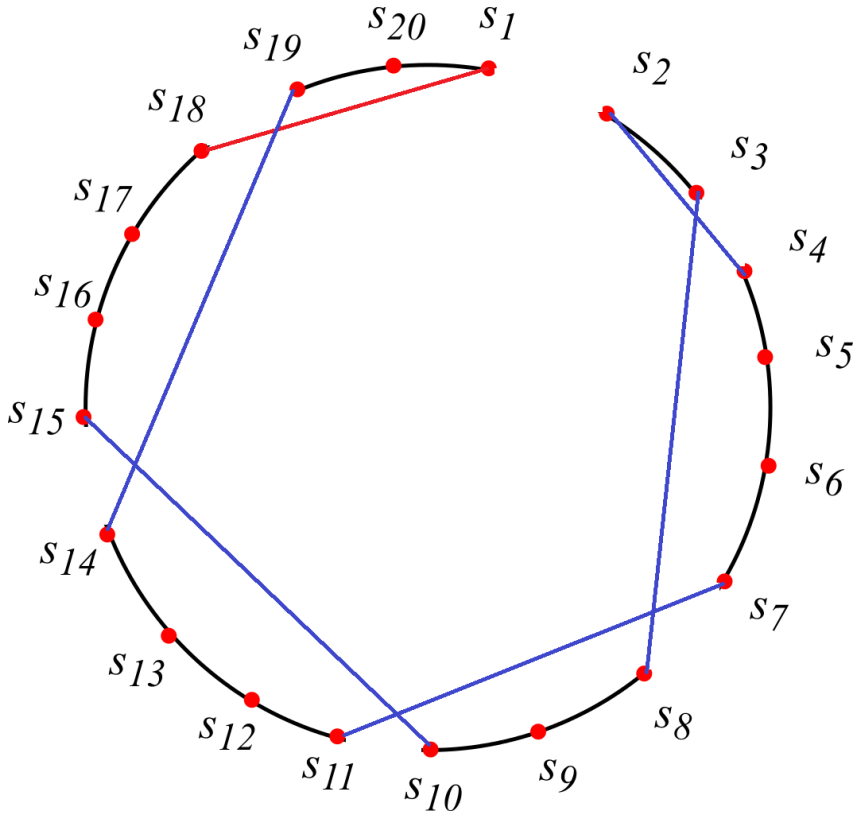


$$x^{iv} = (s_1\ \overline{s_{14}\ s_{13}\ s_{12}\ s_{11}\ s_7\ s_6\ s_5\ s_4\ s_2\ s_3\ s_8\ s_9\ s_{10}\ s_{15}\ s_{16}\ s_{17}\ s_{18}}\ s_{19}\ s_{20})$$

Cerchiamo il miglior scambio che rompa (s_1, s_{14}) e un altro arco di $x \cap x^{iv}$
Supponiamo che sia lo scambio $(1, 18)$, che inverte il tratto $(s_{14}, \ldots, s_{18})$

Esempio: algoritmo di Lin-Kernighan

Lo scambio $(1, 18)$ sostituisce (s_1, s_{14}) e (s_{18}, s_{19}) con (s_1, s_{18}) e (s_{14}, s_{19})



$$x^v = (s_1\ s_{18}\ s_{17}\ s_{16}\ s_{15}\ s_{10}\ s_9\ s_8\ s_3\ s_2\ s_4\ s_5\ s_6\ s_7\ s_{11}\ s_{12}\ s_{13}\ s_{14}\ s_{19}\ s_{20})$$

Cerchiamo il miglior scambio che rompa (s_1, s_{18}) e un altro arco di $x \cap x^v$
Supponiamo che tutti diano soluzioni peggiori di x:
si termina, restituendo la miglior soluzione trovata

Dettagli implementativi

- per evitare di distruggere le mosse già fatte, il secondo arco che viene cancellato ogni volta deve appartenere alla soluzione iniziale
  (*il primo è il primo arco aggiunto dallo scambio precedente*)
- questo implica un limite superiore alla lunghezza della sequenza
- si dimostra che interrompere la sequenza appena gli scambi non migliorano più la soluzione iniziale non danneggia il risultato
  - la variazione complessiva dell'obiettivo è la somma delle variazioni dovute ai singoli scambi

$$\delta f^{(o_1,\ldots,o_k)}(x) = \sum_{\ell=1}^{k} \delta f^{(o_\ell)}(x)$$

  - ogni sequenza di numeri con somma negativa ammette una permutazione ciclica le cui somme parziali sono tutte negative
    (ad esempio, *3,1,-5,2,-3* $\rightarrow$ *-5,2,-3,3,1*)
  - quindi, esiste una permutazione ciclica della sequenza di mosse $(o_1, \ldots, o_k)$ che sia vantaggiosa ad ogni passo

$$\delta f^{(o_1,\ldots,o_k)}(x) < 0 \Rightarrow \exists h : \sum_{\ell=1}^{\ell} \delta f^{(o_{h+1},\ldots,o_{h+\ell})}(x) < 0 \; \forall \ell = 1, \ldots, k$$

Metodi *destroy-and-repair*

Ogni scambio dalla soluzione x alla soluzione x' si può vedere come

- aggiunta a x del sottoinsieme $A = x' \setminus x$
- eliminazione da x del sottoinsieme $D = x \setminus x'$

Aggiunte ed eliminazioni combinate producono scambi, ma

- i sottoinsiemi $x \cup \{j\} \setminus \{i\}$ ottenuti da scambi di elementi singoli potrebbero essere tutti inammissibili o di pessima qualità
- allargare l'intorno a scambi di più elementi può essere inefficiente
- in molti problemi A e D possono avere cardinalità diversa, poiché la cardinalità delle soluzioni non è uniforme (*per es., KP, SCP...*)

Un'alternativa è

- cancellare un sottoinsieme $D \subset x$ di cardinalità $\leq k$ e completarla con un'euristica costruttiva (*specialmente se* $x \setminus D \in X$ *per ogni* D)
- aggiungere un insieme $A \subset B \setminus x$ di cardinalità $\leq k$ e sfrondarla con un'euristica distruttiva (*specialmente se* $x \cup A \in X$ *per ogni* A)

Metodi *destroy-and-repair*

La complessità scende da $O\left(n^{|A|} n^{|D|} \gamma(n)\right)$, dove

- il numero dei possibili sottoinsiemi da aggiungere è $O\left(n^{|A|}\right)$
- il numero dei possibili sottoinsiemi da eliminare è $O\left(n^{|D|}\right)$
- $\gamma(n)$ è la complessità di valutare ammissibilità e obiettivo

ad $O\left(n^{|D|} T_{\text{costr}}(n, |D|)\right)$ o $O\left(n^{|A|} T_{\text{distr}}(n, |A|)\right)$, dove

- $T_{\text{costr}}(n, |D|)$ è la complessità dell'euristica costruttiva
- $T_{\text{distr}}(n, |A|)$ è la complessità dell'euristica distruttiva

La complessità si può ridurre ancora come già descritto

- concentrare gli scambi su elementi di costo basso o alto
- applicare la strategia *first-best* anziché *global-best*

Visita efficiente di intorni esponenziali

Un'altra famiglia di metodi si basa su intorni la cui soluzione ottima si possa trovare risolvendo un problema su un'opportuna matrice o grafo

- packing: *Dynasearch*
- ciclo di costo negativo: scambi ciclici
- cammino minimo: *ejection chains*, *order-and-split*

Tali matrici e grafi ausiliari vengono in genere definiti di miglioramento

Dynasearch

La variazione $\delta f^{(o)}(x)$ della funzione obiettivo a seguito di una mossa elementare $o \in \mathcal{O}$ spesso dipende solo da una parte della soluzione

Operazioni o' che agiscono su altre parti della soluzione hanno un effetto indipendente: non importa l'ordine con cui si eseguono

$$f(o'(o(x))) = f(o(o'(x)))$$

Se $f(\cdot)$ è additiva, l'effetto dei due scambi si somma semplicemente

Esempio:

- scambi fra rami diversi per il *CMSTP* o cammini diversi per il *VRP*
- scambi 2-opt per il *TSP* che operano su tratti disgiunti del ciclo

Dynasearch

La mossa composta è un insieme di mosse elementari come nella *VDS*
Però le mosse elementari devono avere effetti mutuamente indipendenti sull'ammissibilità e sull'obiettivo

La situazione si può modellare con una matrice di miglioramento A in cui

- le righe rappresentano le componenti della soluzione
  (per es., rami nel *CMSTP*, percorsi nel *VRP*, tratti di ciclo nel *TSP*)
- le colonne rappresentano le mosse elementari:
  ogni colonna ha un valore pari al miglioramento $-\delta f$ dell'obiettivo
- se la mossa j impatta sulla componente i, $a_{ij} = 1$; altrimenti $a_{ij} = 0$

Si vuole determinare l'impaccamento ottimo delle colonne, cioè il sottoinsieme di colonne di valore massimo che non coprano la stessa riga

Il *Set Packing Problem* è in genere $\mathcal{NP}$-difficile, ma

- su particolari matrici è polinomiale (è il caso del *TSP*)
- se ogni mossa tocca al massimo due componenti
  - le righe diventano nodi
  - le colonne diventano lati
  - ogni impaccamento di colonne diventa un accoppiamento

  dunque un problema di accoppiamento massimo, che è polinomiale

Scambi ciclici

In molti problemi

- le soluzioni ammissibili partizionano gli elementi in componenti S_ℓ (*i vertici o i lati in rami per il CMSTP, i nodi o gli archi in cammini per il VRP, gli oggetti fra i contenitori nel BPP, ecc...*)
- l'ammissibilità è associata alle singole componenti
- la funzione obiettivo è additiva rispetto alle componenti

$$f(x) = \sum_{\ell=1}^{r} f(S_\ell)$$

È naturale definire per questi problemi l'insieme di operazioni $\mathcal{T}_k$ che contiene i trasferimenti di k elementi dalla propria componente a un'altra
Da $\mathcal{T}_k$ deriva il relativo intorno $N_{\mathcal{T}_k}$

- spesso i vincoli di ammissibilità ostacolano i trasferimenti semplici
- ma il numero dei trasferimenti multipli cresce rapidamente con k

Vogliamo trovare un sottoinsieme di $N_{\mathcal{T}_k}$ efficientemente esplorabile

Il grafo di miglioramento

Il grafo di miglioramento permette di descrivere sequenze di trasferimenti

- un nodo i corrisponde a un elemento i dell'insieme base B
- un arco (i, j) corrisponde al
  - trasferimento dell'elemento i dalla sua componente attuale S_i alla componente attuale S_j dell'elemento j
  - eliminazione dell'elemento j
- il costo dell'arco c_{ij} corrisponde alla variazione della componente dell'obiettivo associata a S_j

$$c_{ij} = f(S_j \cup \{i\} \setminus \{j\}) - f(S_j)$$

con $c_{ij} = +\infty$ se è inammissibile trasferire i eliminando j

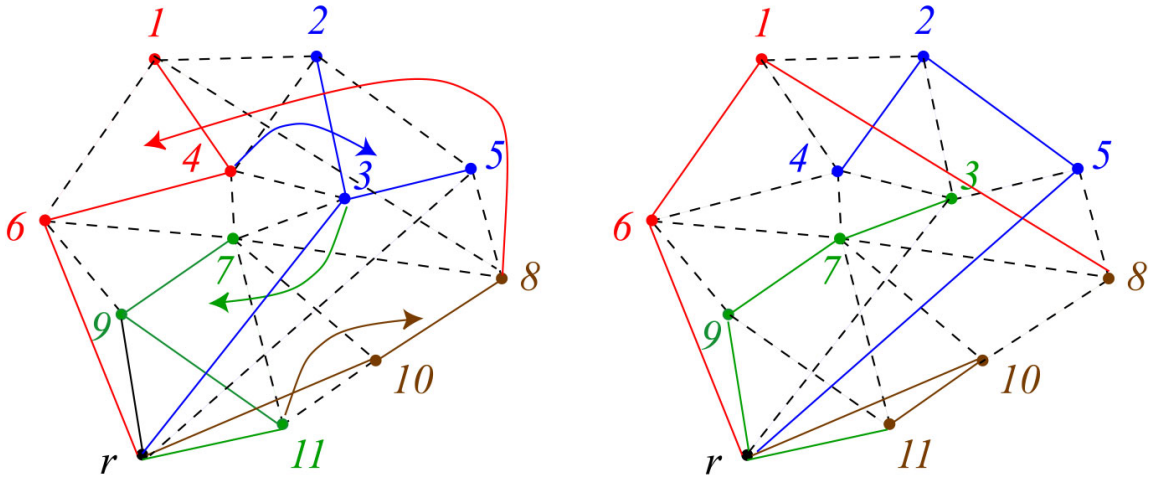
Un ciclo in tale grafo corrisponde a una sequenza chiusa di trasferimenti

Il costo del ciclo corrisponde al costo della sequenza

- ma solo se ogni nodo sta in una diversa componente

Cerchiamo il ciclo di costo minimo che soddisfi questa condizione

Esempio: il *CMSTP*



(4,*Rosso*,*Blu*) (3,*Blu*,*Verde*) (11,*Verde*,*Marrone*) (8,*Marrone*,*Rosso*)

Mossa composta:

- il vertice 4 passa dal ramo rosso al ramo blu e scaccia il vertice 3
- il vertice 3 passa dal ramo blu al ramo verde e scaccia il vertice 11
- il vertice 11 passa dal ramo verde al ramo marrone e scaccia il vertice 8
- il vertice 8 passa dal ramo marrone al ramo rosso e scaccia il vertice 4

I pesi dei vari rami rimangono identici (se $w_v = 1$) o simili
Quindi la soluzione è certamente (o probabilmente) ammissibile

Ricerca del ciclo di costo minimo

Il problema è in realtà $\mathcal{NP}$-difficile, ma

- il vincolo di dover toccare una volta sola ogni componente permette algoritmi di programmazione dinamica abbastanza efficienti
  (*se le componenti sono r, il ciclo ha al massimo r archi*)
- una sequenza di numeri con somma negativa ammette sempre una permutazione ciclica con somme parziali tutte negative
  (ad esempio, *3,1,-5,2,-3* $\rightarrow$ *-5,2,-3,3,1*);
  quindi si possono scartare tutti i percorsi parziali di costo ≥ 0

Inoltre esistono algoritmi polinomiali per calcolare

- cicli negativi qualsiasi (Floyd-Warshall)
- cicli di costo medio minimo (costo totale diviso il numero degli archi)

Questi cicli, pur non rispettando il vincolo sulle componenti, forniscono

- una stima per difetto che può dimostrare l'inesistenza di cicli negativi
- un ciclo negativo che può rispettare il vincolo per fortuna oppure essere modificato fino ad ottenerne uno

Infine, esistono algoritmi polinomiali euristici

Ejection chains: catene di scambi non cicliche

È anche possibile creare catene di trasferimenti non cicliche, ma aperte, in modo che la cardinalità delle componenti possa variare

- una componente perde un elemento
- zero o più componenti perdono un elemento e ne acquistano un altro
- una componente acquista un elemento

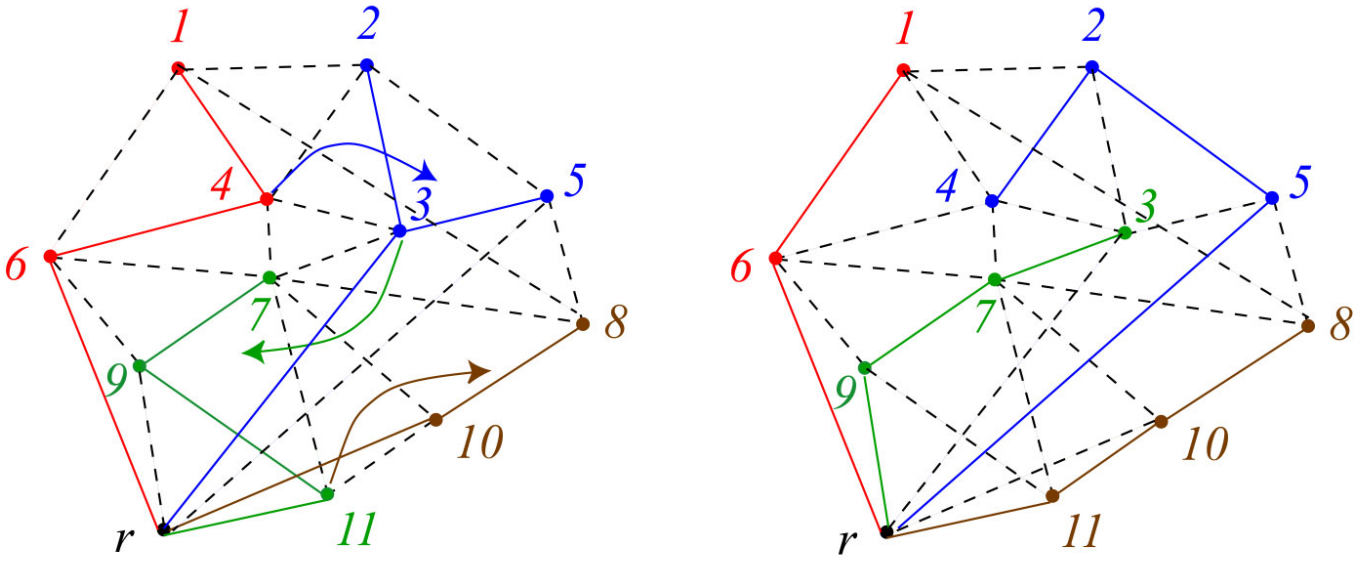
Basta aggiungere al grafo di miglioramento

- un nodo sorgente
- un nodo per ogni componente
- archi dal nodo sorgente ai nodi associati agli elementi
- archi dai nodi associati agli elementi ai nodi associati alle componenti

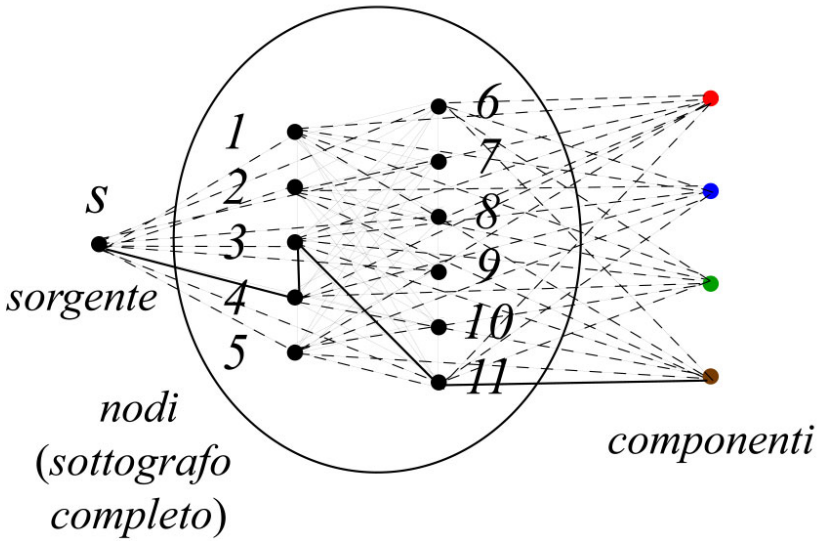
A questo punto, si cerca il cammino di costo minimo che

- vada dal nodo sorgente a un nodo componente
- tocchi un solo nodo componente (la topologia lo garantisce)
- non tocchi mai più di un nodo associato a ciascuna componente (considerando sia i nodi elemento sia i nodi componente)

Esempio: il *CMSTP*



(4,*Rosso*,*Blu*) (3,*Blu*,*Verde*) (11,*Verde*,*Marrone*)



Order-first split-second

Il metodo *Order-first split-second* per i problemi di partizione

- costruisce una permutazione iniziale degli elementi
- partiziona gli elementi in componenti in modo ottimo sotto il vincolo aggiuntivo che elementi della stessa componente siano consecutivi nella permutazione iniziale

Ovviamente, la soluzione dipende dalla permutazione iniziale:
è ragionevole ripetere la soluzione per diverse permutazioni creando un metodo a due livelli

1. al livello superiore, si modificano la permutazione
2. al livello inferiore, si ottimizza la partizione data la permutazione

Problema: poiché le permutazioni sono più numerose delle soluzioni, non è detto che cambiando permutazione cambi la soluzione

Il grafo ausiliario

Anche in questo caso si sfrutta un grafo ausiliario

Data la permutazione $(s_1, \ldots, s_n)$ degli elementi dell'insieme base E

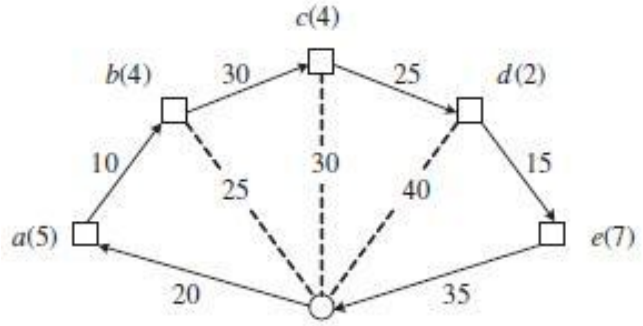
- ogni nodo s_i corrisponde a un elemento s_i dell'insieme base E, più un nodo s_0 fittizio
- ogni arco (s_i, s_j) con $i < j$ corrisponde a una componente potenziale $S_\ell = (s_{i+1}, \ldots, s_j)$ formata dagli elementi della permutazione
  - da s_i escluso
  - a s_j compreso
- il costo c_{s_i, s_j} corrisponde al costo della componente $f(S_\ell)$
- l'arco non esiste se la componente è inammissibile

Di conseguenza

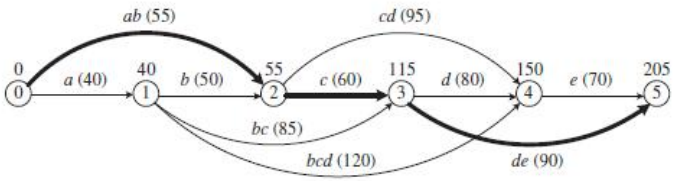
- ogni cammino da s_1 a s_n rappresenta una partizione di B
- il costo del cammino coincide con il costo della partizione
- il grafo è aciclico: trovare il cammino ottimo costa $O(m)$ dove $m \leq n(n-1)/2$ è il numero degli archi

Esempio: il *VRP*

Data un'istanza del *VRP* con 5 nodi e capacità $W = 10$



gli archi corrispondenti a percorsi inammissibili (peso $> W$) non esistono, i costi sono i costi dei percorsi ciclici $(d, s_{i+1}, \ldots, s_j)$



Il percorso ottimo corrisponde a tre cicli: (d, s_1, s_2, d), (d, s_3, d) e (d, s_4, s_5, d)