La programmazione lineare è uno strumento di programmazione matematica che consente la modellizzazione e risoluzione di problemi di ottimizzazione.

**Funzione obbiettivo**: stabilisce se bisogna massimizzare o minimizzare una determinata funzione.

**Vincoli:** stabiliscono la scarsità di una risorsa e la funzione obbiettivo deve rispettare quest’ultimi

Cambiando i vincoli è possibile valutare scenari differenti

**La programmazione lineare usa SOLO funzioni lineari, esponente =1 o 0 se è una costante, altrimenti si parla di programmazione non lineare.**

Nei vincoli i segni ammessi sono solo >= <= e =

*Se si utilizzassero le disuguaglianze di tipo ">", "<" e "≠", se usassi > e volessi minimizzare non avrei un valore minimo, stessa cosa con < se volessi massimizzare. Con il diverso si ci ricollega ai casi precedenti se dico x>=5 e x!=5 avrei x>5 e si presenta nuovamente il problema precedente.*

**Scaletta di modellizzazione:**

* **Capire la decisione e le variabili decisionali**: (in questo caso quanto produrre per ogni unità, **massimizzando** il profitto) x1, x2, x3,x4,x5
* **Qual è la funzione obiettivo?** (**max** 550x1+600x2+350x3+400x4+200x5)
* **Quali sono i vincoli?** **(**numero di macchine, numero di operai, tempo di lavoro)
  + 3 macchine di grinding lavora 2 turni da 8 ore per 6 giorni = **8\*2\*6\*3= ore di grinding a disposizione =288 ore grinding, posso usare al massimo 288 ore per li grinding**

**12x1+20x2+25x4+15x5<=288 🡪 capacità di tempo del grinding.**

* + 2 macchine di drilling lavorano 2 turni da 8 ore per 6 giorni = **8\*2\*6\*2= ore di drilling** **a disposizione =192 ore drilling, posso usare al massimo 288 ore per li drilling**

**10x1+8x2+16x3<=192 🡪 capacità di tempo del drilling.**

* + 8 impiegati che hanno un turno da 8 ore per 6 giorni, 8\*6\*8=384, **20x1+20x2+20x3+20x4+20x5<=384, 20 sono le ore di assemblaggio**
  + **Xn>=0 n=1…5**

**Immagine che contiene lettera

Descrizione generata automaticamente**

Le funzioni obbiettivo se sono più di una possono andare in contrasto tra di loro

**Non tutti i problemi di ottimizzazione possono essere risolti linearmente, ciò impone dei limiti di classe di problemi,**

**Integer Linear Programming (ILP)**È simile alla programmazione lineare, ma alcune variabili **devono** assumere valori interi.

Molti problemi di programmazione lineare intera sono NP completi. Si introducono vincoli non lineare come quelli quadratici. **In sintesi, ILP è più difficile di LP**

**Massimizzare il minimo o Minimizzare il massimo, trasformare funzioni non lineari in lineari se esprimibili come somma di variabili di grado 1**

Si pone la funzione “interna” se min max (x1,x2) prendiamo max e successivamente poniamo gli argomenti come vincoli in base alla funzione. In questo caso z>=x1 e z>=x2 così da avere **min z** come funzione obbiettivo.

La stessa cosa si applica per le funzioni non lineari, solo se sono esprimibili come somma di variabili di grando 1. Come valore assoluto esprimibili come z=|y-d| avremmo z>=(y-d) e z>=(-y+d)Immagine che contiene diagramma

Descrizione generata automaticamente

Variabile di slack: indica ciò che manca, se negativo, o è in eccesso, se positivo, rispetto al valore obbiettivo.

Le variabili di slack vanno sottratte.

**Problemi di flusso**

I problemi di massimo flusso sono riconducibili a problemi di programmazione lineare.

Ci sono devi vincoli:

Gli archi sono orientati e le capacità sono comprese tra 0 e la capacità massima.

Legge di conservazione del flusso, flusso entrante = flusso uscente, il nodo sorgente e destinazione non viene applicata la legge di conservazione perché il flusso esce o entra e basta.

**La soluzione ottima è sempre intere**. Esistono soluzioni frazionarie ammissibili, ma esiste sicuramente una soluzione intera e una sarà quella ottima.

**Matching e Grafo bipartito**

Il grafo bipartito è un caso particolare di matching, dove i nodi sono connessi solo con nodi del lato opposto

Si vuole cercare il massimo matching

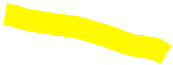
Ponendo il flusso a 1 si ha una sola scelta di direzione (funzione come una variabile binaria)

Nei problemi di flusso si possono modellizzare problemi interi senza imporre vincoli di interezza.

Essendo modellati come problemi di programmazione lineare si ottiene a gratis una soluzione intera.

Immagine che contiene diagramma

Descrizione generata automaticamente



Una volta trovato il flusso massimo, si è trovato anche il matching ottimo prendendo la parte di rete che riguarda il matching



Bisogna indirizzare il flusso sugli archi scarichi e, nel caso del nodo centrale non possiamo andare in alto perché il nodo successivo ha già l’arco saturo, quindi prendiamo quello che va verso il basso

Immagine che contiene grafico

Descrizione generata automaticamente



**Edge Disjoint Paths**

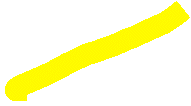
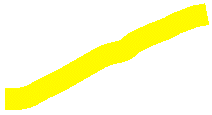
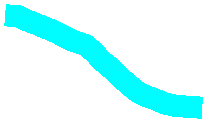
Indice di robustezza della rete, indica il numero di percorsi che hanno dei nodi in comune , ma non hanno archi in comune.

Per calcolare i percorsi strutturiamo il problema come un problema di flusso.

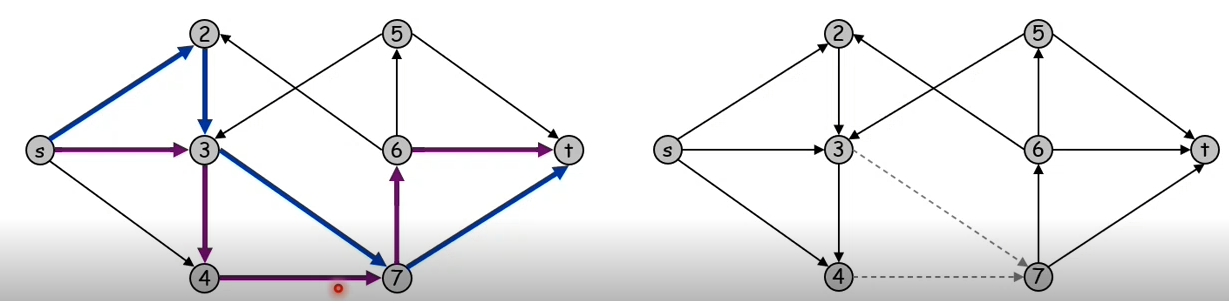
Ponendo le capacità pari a 1 possiamo sfruttare la proprietà di interezza possiamo calcolare il percorso che non dovrà avere archi in comune con altri percorsi.

Immagine che contiene grafico, diagramma

Descrizione generata automaticamente



Il numero di edge disjoint path è pari al numero minimo di archi la cui rimozione disconnetterebbe s con t



**Circulation problem**

Flusso entrante – flusso uscente =d

Dove d = 0 nulla si crea nulla si distrugge

d>0 assorbe il flusso (demand node)

d<0 il nodo sta producendo (supplier node)

il valore se negativo fornisce flusso aggiuntivo se in entrata non è uguale a quello che esce.

Si può riportare a un problema di flusso connettendo degli archi ai nodi supplier con capacità pari alla capacità rimanente.

Ai demand invece si fa uscire un arco di capacità pari a quello che richiede

Immagine che contiene grafico

Descrizione generata automaticamente

Se non si riesce a risolvere il problema di circulation problem devo poter avere un flusso massimo pari al valore uscente da s. il flusso che va ai supply è uguale a quello che ricevono i demand

Immagine che contiene diagramma

Descrizione generata automaticamente

Il primo valore impone il flusso minimo che deve passare

Per ricondurre a un problema di flusso la capacità minima la facciamo assorbire quindi abbiamo v che diventa un demand e w è un supply e nel canale passa upper bound - lowbound quindi 9-2 = 7

Nei problemi a variabili binarie sommare gli elementi e deve essere >=1 per esistere.

**Problemi decisionali**

Si utilizzano variabili booleane

Si fa una sommatoria che ogni elemento soggetto alla condizione sommo il valore della variabile booleana

Il risultato mi restituisce il numero di elementi che ottengo e se è stati preso un elemento se >=1.

**Costruire un “if then”**

If x>0 then y=1

Si traduce nel seguente modo, dobbiamo cercare una relazione tra x e y

X-M\*Y<=0

M è un valore grande, esiste perché se scrivessimo x<=y funzionerebbe solo per valori compresi tra 0 e 1

Si possono imporre dei vincoli soft ovvero variabili che se non rispettano una condizione pagano uno “penale” ovvero peggiora il risultato.

Sono dette variabili indicatore

Moltiplico la parte variabile pe runa variabile indicatrice Y.

Se non si rispetta la condizione l’indicatore va a 1 e quindi va sommato a quello fisso

Xa+Xb<=10+5\*Y

Immagine che contiene diagramma, testo

Descrizione generata automaticamente

Nel sudoku si pongono vincoli Xnumero,riga,colonna per tutte le righe, colonne, blocchi per tutti i numeri e devono essere = 1.

Non c’è una funzione obbiettivo poiché non necessaria dato che dobbiamo solo rispettare i vincoli e ci sono più di una soluzione possibile

Immagine che contiene diagramma, schematico

Descrizione generata automaticamente

**Metodo del Simplesso**

Serve nella risoluzione dei problemi di programmazione lineare.

Se esiste una soluzione ottima finita, esiste sempre una soluzione ottima che giace sui vertici della nostra figura, generata dai vincoli.

La figura è detta poiliedro

L’ottimo del metodo del simplesso è un ottimo globale che corrisponde a un ottimo locale.

Immagine che contiene grafico

Descrizione generata automaticamente

Il risultato del simplesso è uno o più vertici del poliedro, se dovessi massimizzare la terza figura avrei infinito come risultato, **a livello matematico ha senso a livello pratico ci siamo dimenticati un vincolo**.

Possibili soluzioni:

* **Soluzione ottima univoca (vertice)**
* **Soluzione ottima alternata (le soluzioni sono su un lato)**
* **Non ci sono soluzioni ammissibile**
* **Soluzione ottima infinita**

Immagine che contiene testo, antenna

Descrizione generata automaticamente

Riscrivo il vincolo aggiungendo una nuova variabile (di slack) e consente di trasformare una disuguaglianza in un’uguaglianza.

**Immagine che contiene tavolo

Descrizione generata automaticamente**

Immagine che contiene diagramma, schematico

Descrizione generata automaticamente

**Viola e rosso sono equivalenti!!!**

Primo passaggio scrivere il sistema di disequazioni in un sistema di equazioni.

Immagine che contiene diagramma

Descrizione generata automaticamente

Immagine che contiene diagramma

Descrizione generata automaticamente

Immagine che contiene diagramma

Descrizione generata automaticamente

Immagine che contiene diagramma

Descrizione generata automaticamente

Immagine che contiene diagramma

Descrizione generata automaticamente

**Scaletta**

* Trasformare le disuguaglianze in uguaglianze
* (Formato del dizionario, ciò che è a primo membro compare solo a primo membro 1 sola volta) Pongo le variabili di slack(nuove variabili aggiunte) a primo membro e il resto a secondo membro
* Soluzione Associata: pongo a 0 le variabili della funzione obbiettivo (x1 e x2)
* Continuo fin quando la funzione obbiettivo non è più migliorabile (se ho tutti – e una costante può essere un punto se si deve massimizzare)

Il formato del dizionario dice che ciò che compare a destra compare solo a destra quello che è a sinistra compare solo a sinistra.

Metodo 2 fasi si applica quando non è ammassa una soluzione pari a 0

**Branch and Bound**

Immagine che contiene diagramma

Descrizione generata automaticamente

Immagine che contiene diagramma

Descrizione generata automaticamente

Immagine che contiene diagramma

Descrizione generata automaticamente

Si utilizza per risolvere problemi di programmazione lineare intera

Le soluzioni non si trovano sui vertici del politopo

**Branching**: Aggiungo vincoli per eliminare vincoli frazionari

**Problema rilassato**: problema senza vincolo di interezza

Chiusura per integralità: non trovo altre soluzioni candidate

Con il branching si va a creare un albero decisionale

Con il vincolo di interezza le soluzioni possono essere peggiori o uguali della/alla soluzione non vincolata.

Scaletta:

* Risolvo il problema di programmazione lineare non intera, rimuovo il vincolo di interezza
* Aggiungo i vincoli che eliminano la frazionalità della soluzione:
  + Si avranno 2 vincoli

Scelgo una variabile in modo arbitrario per fare branching

I figli ereditano i vincoli del padre

Se ho una soluzione intera mi fermo

La scelta delle variabili cambia nodo per nodo, non necessariamente devo scegliere la stessa a patto che sia frazionaria e non intera.

Criteri di chiusura:

* **Infeasibility**: non c’è una soluzione
* **Interezza**: la soluzione è intera
* **Non ottimalità:** valore non ottimo non migliorabile, le soluzioni successive saranno sempre peggiori o uguali

**Algoritmi**

Tipologie di algoritmi:

* Algo esatti
* Algo approssimati con metodi euristici
* Metaeuristiche, euristiche iterative, non si hanno garanzie di bontà e ottimalità

Algoritmi approssimati:

sono algoritmi con complessità polinomiale

su può calcolare la soluzione nel caso pessimo

Euristiche

Sono degli algoritmi approssimati, ma non è possibile calcolare la qualità della soluzione

**Immagine che contiene diagramma

Descrizione generata automaticamente**



**Trial and error**

Si parte da una soluzione iniziale e iterativamente si modifica al fine di avere una soluzione diversa possibilmente migliore.

Ricerca locale

**TSP e MST**

La relazione tra tsp e mst è che si deve avere un percorso che passa per tutti i nodi con il costo minore.

1\*MST è il **lowerbound** del tsp

Il **lowerbound** lo utilizzo per costruire l’euristisca

**upperbound** Tsp=2\*mst=

**Local Search**

Se non esiste un lowerbound si applica un metodo sperimentale per valutare il lowerbound

Sono alla base delle metaeuristiche

La ricerca locale parte da una soluzione ammissibile e ha lo scopo di migliorarla.

Modificando la soluzione si andrà ad ottenere una soluzione vicina alla soluzione di partenza.

L’insieme dei vicini è detto vicinato, questa operazione è chiamato neighborfunction.

Ci si ferma quando si ci trova in un minimo/massimo locale.

Non esiste un modo univoco per generare il neighbor

**First improvement:** l’esplorazione termina non appena la soluzione è migliore di quella corrente

**Steepset descent:** se trovo un migliorante continuo a cercare nel vicinato, alla fine valuto le soluzioni

Swap di elementi per cambiare la soluzione

La soluzione iniziale di local search DEVE essere ammissibile

Nella ricerca locale si ha il difetto di rimanere bloccati in minimi e massimi locali, potrebbero esserci soluzioni migliori, ma per come funziona la local search si ferma nei minimi/massimi locali, la soluzione sono le metaeuristiche.

Gli ottimi locali sono solo peggiori dell’ottimo globale

**Metaeuristiche**

**Multi start local search**: genera più soluzioni iniziali e applica la local-search

**Simulating annealing:** ha un meccanismo probabilistico e accetta solo le soluzioni miglioranti, la probabilità di prendere soluzioni che peggiorano la soluzione inizialmente è alta, ma andando avanti tenderà a 0; accettando così solo soluzioni miglioranti su cui fare la local search.

Le soluzioni tenderanno a 0 se sono vicine o uguali

If(f(y)<=f(x))

Then x:=y

Else if

Then x:=y

Il parametro c è detto temperatura che all’inizio è molto grande, viene ridotto ad ogni iterazione

**La probabilità di accettare soluzioni peggioranti dipende dalla temperatura e dalla qualità della soluzione**

Il parametro L indica il tempo di esecuzione

**Tabù search**

Accetta vicini non miglioranti per uscire dai min/max locali

Ha una memoria a breve termine che proibisce gli elementi una volta aggiunti

La dimensione della tabella è finita

L'algoritmo prende il nome dal concetto di "taboo" o "tabù", che indica una restrizione imposta alla ricerca di soluzioni simili a quelle già esplorate in precedenza. In altre parole, l'algoritmo tiene traccia di soluzioni che sono già state esplorate e le etichetta come "tabù" per un certo numero di iterazioni, impedendo così che la ricerca torni su di esse.

Nel caso del tsp swappo gli elementi ad ogni iterazione e cerco la migliore soluzione non tabù

Immagine che contiene diagramma

Descrizione generata automaticamente

Ricordarsi di farlo in modo ciclico il primo e l’ultimo elemento sono uguali

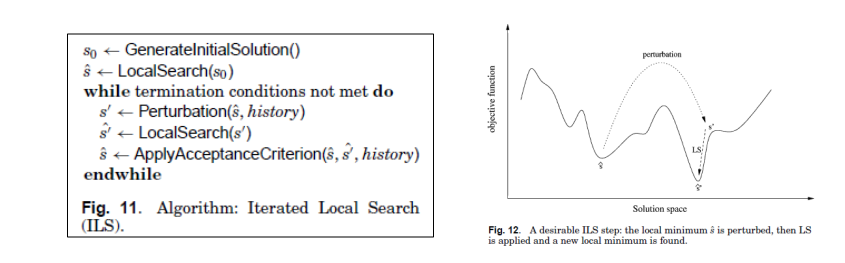
Se un elemento è tabù non considero lo swap (AC e CA sono uguali)

Una volta usciti dalla tabella tabù la soluzione può essere ripresa.

Se una soluzione tabù migliora la migliore soluzione trovata allora si fa un’eccezione e viene aggiunta alla tabù list..

**Local search iterativa**

è una tecnica di ottimizzazione euristica che combina una ricerca locale con un processo di perturbazione e successiva ricerca locale.

perturbare una soluzione significa modificare la soluzione attuale. è un multi-start più conservativa

**variable neighborhood search**

cambia la funzione di neighborhood, ad esempio aumentando la dimensione del neighborhood

aumentando/riducendo il neighborhood aumento la probabilità di avere un punto di minimo/massimo ottimo migliore.

**Guided Local search**

Va a cambiare la funzione obbiettivo

Si potrebbe andare a perturbare dei dati del problema

L’idea è quella di rendere la soluzione attuale non attrattiva.

**Trajectory**: passa da una soluzione ad un'altra, una volta scelta la soluzione ciclo

**Population based**: considera più soluzioni contemporaneamente ottenendo una popolazione di soluzioni, calcolo parallelo

**Algoritmi genetici**

Sono tecniche basata sulla popolazione.

Gli algoritmi consentono la fusione di soluzioni esistenti, mantenendo l’ordine degli elementi.

Si possono effettuare anche mutazioni delle soluzioni.

Le nuove generazioni rimpiazzano le vecchie generazioni generatrici.

**Ant colony optimization**

Le modifiche inizialmente sono casuali, quelle con maggiore probabilità di scelta verranno seguite

**Intensificazione e Diversificazione**

Intensificazione: partendo da una soluzione si modifica per migliorare ulteriormente

Diversificazione: Si modifica in modo pesante la soluzione per migliorare la soluzione

**Greedy Algorithms**

**Load Balancing**

**List scheduling**: algoritmo greedy che cerca la macchina più scarica per avere il carico bilanciato, non sempre dà la soluzione ottima. La soluzione è 2 volte la soluzione ottima

**Lowerbound**

Il valore ottimo è maggiore o uguale del massimo dei tempi di processamento

L’ottimo deve essere maggiore della media dei tempi dei jobs

LPT Rule ordino i jobs in base al tempo, processo prima i più lunghi e ottiene 4/3 rispetto all’ottimo.

**Interval scheduling**

Obbietto: trovare il maggiore sottoinsieme di task compatibili

Earliest finishing time, è ottima, cerca il punto in cui differisce da quella ottima ed effettua swap(scambio) con la soluzione ottima nel punto in cui differisce, così che ha un valore più vicino a quella ottima.

**Computational complexity**

**P=NP**

Ci focalizziamo su problemi decisionali.

Un problema decisionale è un problema che ha come risultato o vero o falso.

Gli input e gli output sono codificati usando un alfabeto (e.g 0,1)

La classe P è una classe di problemi decisionali risolvibili in tempo polinomiale.

NP=non-deterministic polynomial time

**NP è la classe dei problemi di decisione per i quali esiste un certificatore in tempo reale.**

**Un certificatore C(s,t) è un algoritmo che ha come input una stringa e a dimostrazione, verifica se una stringa appartiene al dominio del problema in base alla dimostrazione.**

**NP appartiene alla classe esponenziale e P è contenuto in NP**

**TSP**: è ESP, ma è anche NP

Non è stato ancora dimostrato che tutti i problemi NP sono risolvibili in P

Un problema Y è NP-completo con la proprietà che se posso risolverlo in PTIME allora posso risolvere qualsiasi problema in NP in PTIME

**Circuit satisfiability**

Circuit satisfability è np completo, è un circuito logico (input {0,1})

**Risolvere circuiti satisfiability in tempo polinomiale, consentirebbe di trovare la t ovvero l’input e dimostrare che P=NP**

**3 sat è complesso come circuit sat**

**Polinomial time reduction**

Riduzione di un problema ad un altro in tempo polinomiale

Immagine che contiene diagramma

Descrizione generata automaticamente

**Computabilità**

quali compiti computazionali possono essere risolti

**Complessità**

quali compiti computazionali possono essere risolti entro una certa quantità di risorse computazionali