Corso di Metodi e Modelli per l'Ottimizzazione Combinatoria - Relazione progetto

Universitá degli studi di Padova
Anno Accademico 2015/2016

Studente: Giacomo Quadrio
Matricola: 1061566

Sommario

Il progetto del corso di Metodi e Modelli per l'Ottimizzazione Combinatoria consiste nel risolvere un problema che vede coinvolta un'azienda metalmeccanica che produce pannelli forati per la costruzione di quadri elettrici. La foratura di questi è eseguita attraverso un macchinario a controllo numerico dotato di una punta diamantata che, muovendosi sul pannello secondo una sequenza programmata, produce i fori nelle posizioni desiderate. L'obiettivo è quindi quello di individuare la sequenza di foratura ottimale che minimizzi i tempi di produzione, tenendo conto che il tempo necessario per la foratura è lo stesso e costante per tutti i punti. Il cosiddetto Problema del Commesso Viaggiatore, più comunemente noto come TSP (*Travelling Salesman Problem*), in cui è richiesto di trovare il *circuito Hamiltoniano* di costo complessivo minimo dato un grafo G = (V,E) pesato con costi c_{ij} assegnati ad ogni arco (i,j). Un circuito Hamiltoniano non è altro che un ciclo all'interno del grafo tale che ogni vertice v ∈ V appartenga al ciclo e venga percorso una sola volta.

Nei problemi di tipo TSP abbiamo poi che i costi degli archi rispettano la cosiddetta *disuguaglianza triangolare* ovvero dati tre punti A, B e C, la distanza tra A e C deve essere al massimo la distanza tra A e B più la distanza tra B e C. Più nel dettaglio il TSP in esame in questo progetto è un *TSP euclideo* o planare cioè nel quale si usano le ordinarie distanze euclidee per i costi degli archi. Nonostante il problema rimanga NP-difficile molte euristiche riescono a lavorare meglio.

1. Modello del problema

Il problema oggetto del progetto puó essere formulato come un problema di ottimizzazione su reti di flusso partendo quindi da un grafo G = (N,A). Scegliendo arbitrariamente un nodo di partenza 0 ∈ N impostiamo ad |N| il flusso uscente da esso in modo tale che venga spinto verso altri nodi. Tale operazione ha peró dei vincoli ovvero ciascun nodo, eccetto l'origine, riceverá una e una sola unitá di flusso, ogni nodo sia visitato una e una sola volta e che il costo del cammino, in termini di pesi c_{ij}, sia minimo.

1.1. Il modello nella Programmazione Lineare Intera

Il problema può essere formalizzato con il seguente modello di programmazione lineare intera. Avremo quindi:

Insiemi

- **N:** nodi del grafo, rappresentano le posizioni dei fori da realizzare
- **A:** insieme degli archi (i,j) con i e j ∈ N. Essi rappresentano il tragitto per spostarsi dal nodo i al nodo j

Parametri

- $\mathbf{c_{ij}}$**:** tempo impiegato per spostarsi dal nodo i al nodo j con i e j ∈ N e l'arco (i,j) ∈ A.
- **0:** nodo di partenza del cammino ∈ N.

Variabili decisionali

- $\mathbf{x_{ij}}$**:** unità di flusso trasportate da i a j con i e j ∈ N e l'arco (i,j) ∈ A.
- $\mathbf{y_{ij}}$**:** indica l'utilizzo dell'arco (i,j), 1 se viene utilizzato, 0 altrimenti. Avremo che i e j ∈ N e l'arco (i,j) ∈ A.

Vincoli

Il modello, a questo punto, prevede un totale di cinque vincoli differenti che possono essere indicati come segue:

1. Il flusso uscente da x_{0j} deve essere massimo, cioè |N|
2. Ogni nodo utilizza al massimo una unità di flusso, tranne il nodo di partenza
3. Ogni nodo ha un solo arco in entrata
4. Ogni nodo ha un solo arco in uscita
5. Se vi è un'unità di flusso trasportata da i a j deve di conseguenza esserci un arco che va da i a j

2. Metaeuristiche scelte per il modello e loro implementazione

Il progetto da svolgere richiesto dal corso di Metodi e Modelli per l'Ottimizzazione Combinatoria prevede la risoluzione del modello di programmazione lineare intera tramite due tecniche ovvero usando il risolutore IBM CPLEX ed una o più metaeuristiche a nostra scelta. Una volta fatto ciò si procederà testando i metodi con delle istanze di prova ed i risultati e statistiche confrontati tra di loro per valutarne le prestazioni.

Nello specifico, il problema in esame è un problema di ricerca di vicinato e consiste nel definire una soluzione iniziale e cercare di migliorarla esplorando un intorno di questa soluzione; quindi i metodi scelti ed utilizzati all'interno del progetto sono due, la Local Search ed il Simulated Annealing. Si tratta di metodi metaeuristici ovvero tecniche generali di schemi algoritmici concepiti indipendentemente dal problema specifico. Tali metodi definiscono delle componenti e le loro iterazioni al fine di pervenire ad una buona soluzione ed esse inoltre devono essere specializzate per i singoli problemi. In entrambe le metaeuristiche implementate si è infine utilizzato un vicinato del tipo 2-opt.

2.1. ***Descrizione delle metaeuristiche***

2.1.1. ***Local Search***

Si definisce una soluzione iniziale e si cerca di migliorarla esplorando un intorno di questa soluzione. Se l'ottimizzazione sull'intorno della soluzione corrente produce una soluzione migliorante il procedimento viene ripetuto partendo, come soluzione corrente, dalla soluzione appena determinata. L'algoritmo termina quando non è più possibile trovare delle soluzioni miglioranti nell'intorno della soluzione corrente o quando è stata determinata una soluzione con valore della funzione obiettivo uguale a qualche bound.

2.1.2. ***Simulated Annealing***

Il concetto su cui si basa questa metodologia prende spunto dal mondo reale ovvero consiste nel simulare il comportamento di un processo termodinamico di ricottura di materiali solidi. Se un materiale solido viene riscaldato oltre il proprio punto di fusione e poi viene raffreddato in modo da riportarlo allo stato solido, le sue proprietà strutturali dipenderanno fortemente dal processo di raffreddamento. In sostanza tale algoritmo simula il cambiamento di energia di un sistema sottoposto a raffreddamento, fino a che non converge ad uno stato solido; ciò permette di cercare soluzioni ammissibili di problemi di ottimizzazione cercando di convergere verso soluzioni ottime.

2.2. ***Implementazione degli algoritmi***

2.2.1. ***Risolutore IBM CPLEX***

Il risolutore IBM CPLEX è un programma di ottimizzazione che prende il nome dal metodo del simplesso implementato in linguaggio C (anche se ad oggi offre interfacce verso altri linguaggi ed ambienti). Esso trova soluzioni a problemi di programmazione lineare intera anche di notevoli dimensioni, utilizzando le varianti primale o duale del metodo del simplesso. L'implementazione eseguita per il progetto ricalca a grandi linee quanto fatto durante le lezioni di laboratorio: innanzitutto viene letta la matrice delle adiacenze e linearizzata in un array di dimensioni pari a n*n. Dopodiché si inseriranno in coda i costi degli archi c_{ij} come coefficienti delle variabili decisionali y_{ij}. A questo punto dovranno infine essere creati i vari vincoli definiti nel paragrafo 1.1, sotto la voce "Vincoli", attraverso appositi vettori.

2.2.2. ***Local Search***

Codice 1: Local Search

```
while (true){
        vector<int> newSol = findBestN(sol);
        if (evaluate(newSol) >= evaluate(sol)){
                return evaluate(sol);
        }else{
                sol = newSol;
        }
}
```

Come si può vedere, l'algoritmo di Local Search avvia la sua ricerca da una soluzione iniziale sol che gli viene data in input al momento della chiamata del metodo. All'interno del progetto la Local Search è utilizzata in realtà non come metaeuristica principale ma come metaeuristica per la fase di intensificazione della Simulated Annealing, di consequenza le viene data in input la soluzione che quest'ultima ha trovato così da migliorarne il risultato. Nulla toglie però che questa tecnica si possa utilizzare anche individualmente dandole in pasto, in questo secondo caso, una nuova soluzione di partenza tramite il metodo getInitialSol.

Entrando nel dettaglio, l'algoritmo è stato implementato utilizzando un unico ciclo while che opera finché non viene restituito in output un valore. Questo significa che ciò avverrà unicamente quando, dopo aver calcolato una nuova soluzione con newSol = findBestN(sol), il suo valore sarà peggiore del valore della soluzione corrente. Se così non è la soluzione corrente viene aggiornata alla soluzione appena calcolata ed il ciclo continua ad operare fino al punto in cui viene trovata una soluzione peggiorativa.

Di seguito verranno ora elencate nel dettaglio le funzioni chiave utilizzate nella Local Search:

- **findBestN:** Questa funzione, partendo da una soluzione iniziale, calcola una nuova soluzione invertendo gli elementi contenuti all'interno di un intervallo pos1 - pos2 che cresce progressivamente. Quando la nuova soluzione è migliore della vecchia il ciclo termina e restituirà in output tale valore

- **evaluate:** Funzione che serve per calcolare il valore di una soluzione

2.2.3. ***Simulated Annealing***

Codice 2: Simulated Annealing

```
sol = getInitialSol(random);
n_passi = 100000.0 * n / 5;

while (step < n_passi){
        newSol = getNeigh(sol,2,true);
        float de = evaluate(sol) - evaluate(newSol);
        if (de > 0){
                sol = newSol;
```

```
        }else{
                temp = 1-(step/n_passi);
                double prob = exp((de)/temp);
                srand(time(NULL)+for_random);
                for_random = for_random + 1;

                if (prob*100 > (rand()%100) ){
                        sol = newSol;
                }
        }
step ++;
}

return(LocalSearch(sol));
```

Per quanto concerne invece l'algoritmo di Simulated Annealing, esso crea innanzitutto una soluzione iniziale da cui partire tramite la funzione getInitialSol, dopodiché troviamo un ciclo while principale che opererà finché step <= n_passi; da notare che il numero di passi inoltre è dinamico così che cresca al crescere del numero di nodi coinvolti nel problema. In cosa consiste quindi questo algoritmo? Viene calcolata una nuova soluzione attraverso la funzione getNeigh, dopodiché verrà calcolata la differenza tra il valore della soluzione corrente e quello della nuova soluzione. Se il delta ottenuto è maggiore di zero significa che la nuova soluzione è migliorativa e quindi aggiornerò di conseguenza sol, se invece così non è procederemo in maniera differente rispetto alla Local Search. Andremo infatti a calcolare per prima cosa la temperatura di raffreddamento, valore che è coinvolto nel calcolo della probabilità di accettare una mossa peggiorativa. Questa probabilità è calcolata come segue:

$$\text{prob} = \exp(\ \delta/\text{t})$$

dove δ è l'entità del peggioramento de e t è la temperatura temp di raffreddamento. Nel caso la probabilità prob sia maggiore di un numero random calcolato attraverso la funzione srand ciò comporterà appunto l'accettare la mossa peggiorativa, altrimenti essa verrà scartata. Come si modifica però la probabilità p? Essa diminuisce al crescere del peggioramento indotto dalla mossa stessa e cresce al crescere della temperatura t di processo. Tale parametro di controllo viene generalmente inizializzato ad un valore $t_0 > 0$ eleva-

to e aggiornato con una certa frequenza in modo dinamico. Nello specifico abbiamo che la temperatura in questo progetto è stata così calcolata:

$$\text{temp} = 1\text{-}(\text{step}/\text{n\_passi})$$

Inizialmente essa avrà un valore alto poiché il numero di passi sarà molto piccolo ma mano a mano che crescerà la temperatura diminuirà di conseguenza.

Al termine delle operazioni eseguite tramite il ciclo while andremo ad effettuare infine una **fase di intensificazione** tramite l'operazione di Local Search eseguita sulla soluzione migliore calcolata in precedenza. Il motivo di ciò è che la Local Search è una tecnica relativamente economica in quanto a tempi di esecuzione e permette di raffinare ulteriormente la soluzione trovata.

Di seguito verranno ora elencate nel dettaglio le funzioni chiave utilizzate nella Local Search:

- **getInitialSol:** Questa funzione crea una soluzione di partenza casuale attraverso l'utilizzo di indici random ottenuti grazie ad uno specifico metodo denominato srand
- **evaluate:** Funzione che serve per calcolare il valore di una soluzione
- **getNeigh:** Questa funzione, partendo da una soluzione iniziale, calcola una nuova soluzione invertendo gli elementi contenuti all'interno di un intervallo pos1 - pos2 i cui indici sono calcolati casualmente grazie ad un metodo denominato srand.

3. Descrizione dei test effettuati

Per verificare le prestazioni degli algoritmi utilizzati nel progetto sono stati condotti diversi test con un numero ben definito di istanze per differenti tipologie di dataset. Nel dettaglio sono stati utilizzati quattro diversi dataset in cui i punti sono disposti come segue:

- **Distribuzione uniforme** ovvero uniformemente disposti all'interno dello spazio
- In **cluster** e disposti in tre raggruppamenti
- In **circoli** e disposti in tre circonferenze

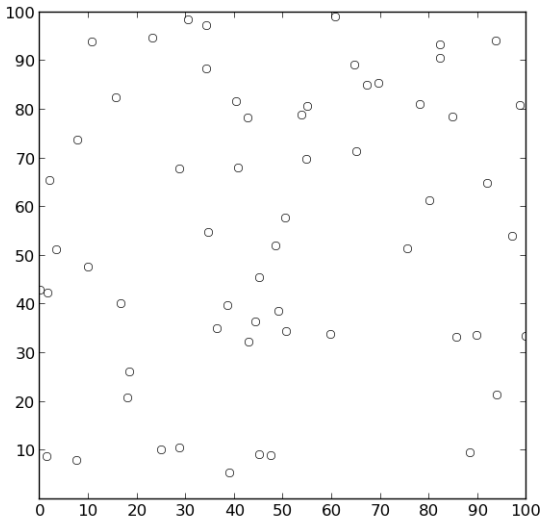

Figura 1: Distribuzione uniforme

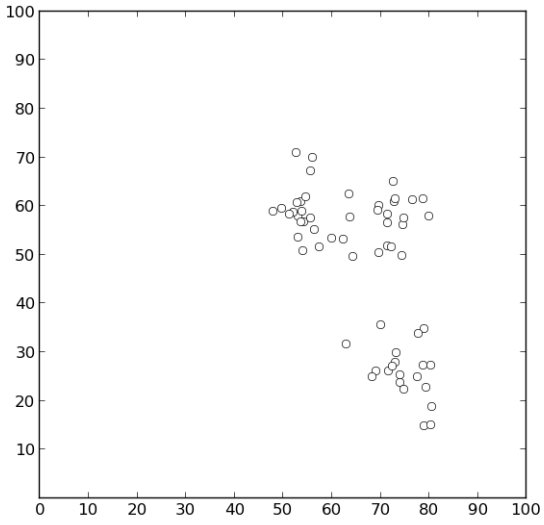

Figura 2: Distribuzione in cluster

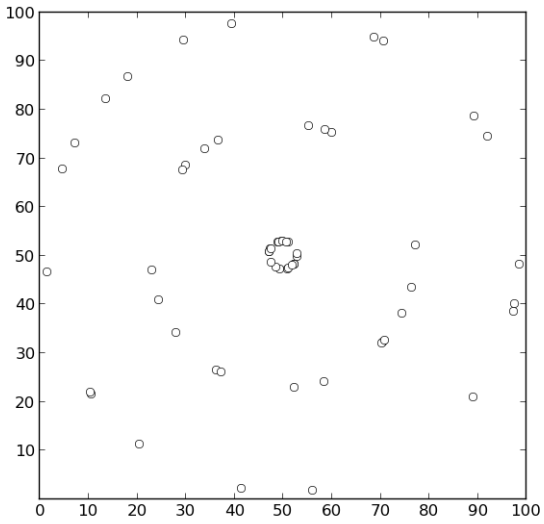

Figura 3: Distribuzione in cerchi

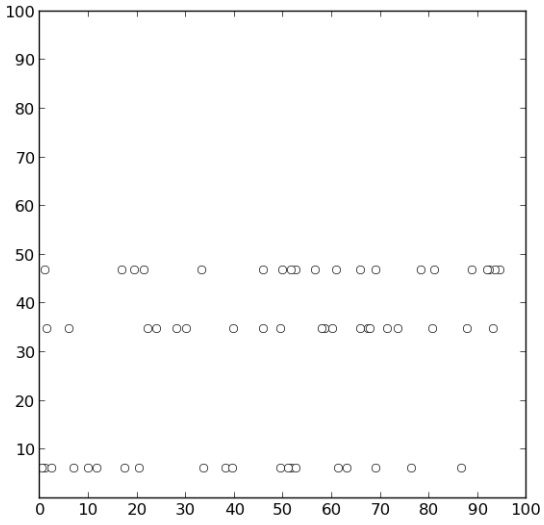

Figura 4: Distribuzione in linee

- In **linea** e disposti in tre linee

Per ciascuna tipologia di dataset sono stati poi selezionati 10 raggruppamenti composti da 4 istanze, ognuno con un numero di nodi che va da 10 a 100 ed in cui la differenza della quantità di elementi tra un gruppo ed il successivo è pari a 10. Le istanze utilizzate sono in realtà matrici delle adiacenze M di dimensioni n*n tali che nella cella m_{ij} vi è la distanza tra il vertice v_i ed il vertice v_j

I test condotti per i vari algoritmi sono stati effettuati utilizzando, di volta in volta, sempre gli stessi dataset così da poter confrontare meglio i risultati. Ogni istanza di ogni raggruppamento di ogni dataset è stata infine data in pasto agli algoritmi per un totale di 10 volte ed i risultati registrati ritenuti interessanti per la valutazione sono stati vari quali il tempo medio e totale impiegato, il valore dell'ottimo, le soluzioni migliori, la varianza e via dicendo.

4. Descrizione dei risultati ottenuti

I risultati completi ottenuti dall'esecuzione degli algoritmi sono stati riportati all'interno dei file Excel denominati "confronti_dati", "cplex_results", "sim_annealing_results" e non quindi direttamente nella relazione a causa della mole di dati che comprendono. In questa sede compariranno quindi solo risultati riassuntivi utili a meglio comprendere le conclusioni a cui si è giunti o, eventualmente, rimandi ai file sopraccitati.

4.1. ***Tempi di esecuzione***

Osservando la figura 5 possiamo constatare un dettaglio davvero interessante: i tempi globali di esecuzione della Simulated Annealing sono inizialmente maggiori per i medesimi dataset rispetto a quelli ottenuti tramite CPLEX, si equiparano per insiemi di 30 punti per poi differenziarsi nuovamente. Dai 30 nodi in poi il grafico in scala logaritmica evidenzia come il CPLEX abbia tempi di esecuzioni ben superiori, con una loro crescita generale molto più rapida rispetto al Simulated Annealing che, invece, ha un andamento molto più moderato. Andiamo però a confrontare alcuni numeri dalla tabella 1:

già a partire dai 60 nodi abbiamo 33,14 secondi di CPLEX contro i 5,45 di Simulated Annealing, una differenza che è destinata a crescere a dismisura con il crescere dei nodi, arrivando a 1418,18 secondi contro gli 11.49 per

Tabella 1: Tempi medi di esecuzione globali

Num Nodi	Tempi CPLEX	Tempi Sim Annealing
10	0,07740625	0,61469375
20	0,3832	1,343675
30	2,42165	2,18094375
40	7,20473125	3,17826875
50	17,98974375	4,20775
60	33,1410875	5,44746875
70	549,3165625	6,76326875
80	568,953975	8,2913125
90	745,47655	9,8671625
100	1418,1834375	11,490575

quanto riguarda 100 elementi; un guadagno per il Simulated Annealing in quest'ultimo caso di un ordine di grandezza di un fattore pari a 1000.

Osserviamo adesso più nel dettaglio i tempi di esecuzione: la figura 6 ribadisce quanto detto fino ad ora ma evidenzia che, se il Simulated Annealing si comporta pressoché nel medesimo modo per ogni tipologia di dataset, il CPLEX invece già a partire da 50 nodi varia notevolmente nei quattro casi. Il dataset in cui abbiamo tempi di esecuzione migliori e più omogenei è quello relativo alla distribuzione uniforme dei punti mentre Cluster e Circle mettono decisamente più in difficoltà il risolutore.

4.2. ***Simulated Annealing: il miglioramento ad opera della Local Search***

Come evidenziato nel paragrafo 2.2, la soluzione ottenuta con l'algoritmo di Simulated Annealing è stata ulteriormente raffinata grazie ad una fase di intensificazione realizzata attraverso una Local Search. Si è optato per tale operazione poiché presenta contenuti tempi di esecuzione in proporzione ai risultati ottenuti.

La figura 7 presenta il grafico globale di miglioramento dei risultati ottenuti dalla Simulated Annealing grazie alla Local Search: ad eccezione dei dataset da 10 punti, dove il guadagno è minimo, in tutti gli altri casi si è avuto un miglioramento della soluzione nell'intervallo 6% - 10%, che ha quindi

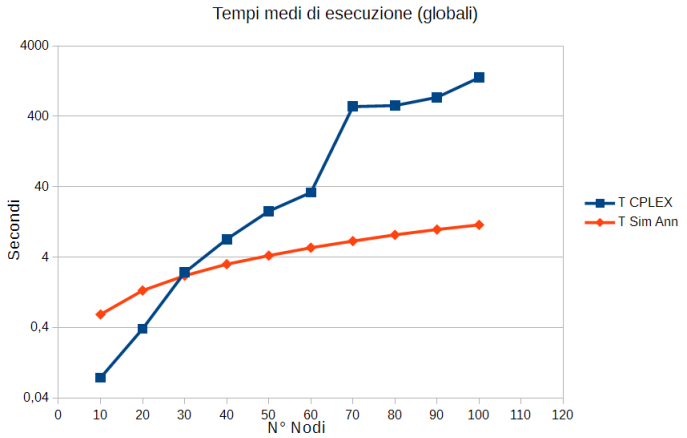

Figura 5: Grafico che riassume i tempi globali di esecuzione registrati per CPLEX e Simulated Annealing

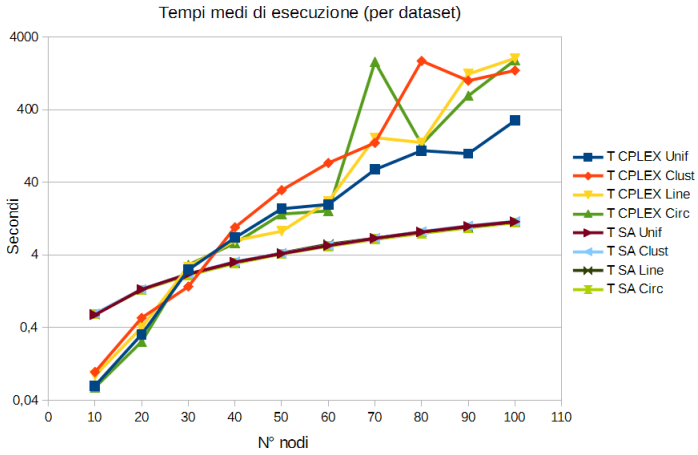

Figura 6: Grafico che riassume i tempi di esecuzione divisi per tipo di dataset registrati per CPLEX e Simulated Annealin

permesso all'algoritmo realizzato per il progetto di avvicinarsi ulteriormente alla soluzione ottima calcolata dal CPLEX. Osservando invece la figura 8 possiamo notare i diversi comportamenti sulle quattro tipologie di dataset utilizzate. Gli andamenti sono in generale molto irregolari ma si possono osservare fluttuazioni più contenute per i dataset a Distribuzione Uniforme e a Cluster mentre la Local Search si è trovata più in difficoltà, ed ha avuto risultati più eterogenei, per quanto riguarda i dataset Line e Circle.

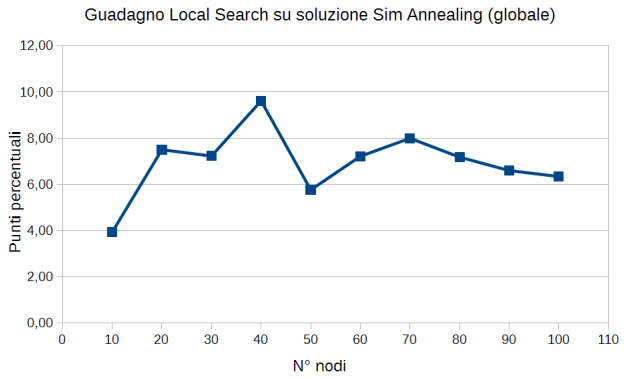

Figura 7: Grafico che riassume il miglioramento globale, in punti percentuali, dei risultati della Simulated Annealing ad opera della Local Search

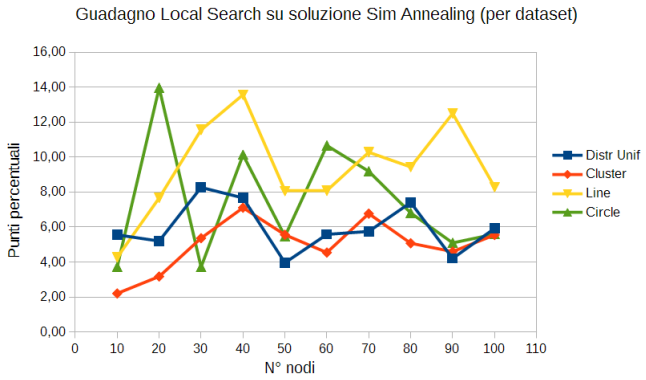

Figura 8: Grafico che riassume il miglioramento per tipologia di dataset, in punti percentuali, dei risultati della Simulated Annealing ad opera della Local Search

4.3. ***Simulated Annealing e CPLEX: soluzioni a confronto***

Se i tempi di esecuzione della Simulated Annealing sono notevolmente migliori rispetto a quelli di CPLEX lo stesso non si può dire per le soluzioni calcolate. L'algoritmo implementato nel progetto infatti trova delle soluzioni che si possono definire sub-ottime, caratteristica tipica dei metodi appartenenti alle metaeuristiche. Nella figura 9 si osserva proprio ciò: l'andamento del peggioramento è crescente ad eccezione dei primi due gruppi di istanze e spazia tra i 0,89 ed i 7,62 punti percentuali in più rispetto all'ottimo di CPLEX da come si può vedere nella tabella sottostante:

Tabella 2: Peggioramento della soluzione ad opera della Sim Annealing

Num Istanze	Peggioramento Sim Annealing
10	100,00%
20	100,05%
30	101,78%
40	100,89%
50	101,06%
60	102,97%
70	107,62%
80	102,17%
90	102,13%
100	102,48%

Questo risultato si è ottenuto nonostante si sia adottato un numero di passi per il Simulated Annealing dinamico, ovvero che cresce al crescere dei punti coinvolti nel problema in esame. Per porre rimedio a ciò un'ipotesi potrebbe essere quella di aumentare ulteriormente il numero di passi eseguiti al crescere dei nodi coinvolti, ma siamo davvero sicuri che in termini di efficienza ciò sia vantaggioso? Nella fase di test si è effettivamente provato ad utilizzare un numero di passi dinamico superiore rispetto alla formula:

$$\text{n\_passi} = 100000.0 * \text{n} / 5$$

ma il guadagno in quanto a miglioramento della soluzione non era proporzionale all'aumento dei tempi di esecuzione, che raggiungevano addirittura un picco di ben 80 secondi in più rispetto ai circa 12 per i gruppi da 100 nodi.

La figura 10 mostra invece il medesimo grafico ma suddiviso per tipi di dataset. In questo caso possiamo osservare che Distribuzione Uniforme e Line hanno un andamento crescente molto simile nel peggioramento da parte della Simulated Annealing della soluzione, mentre Cluster e Circle hanno fluttuazioni talvolta anche importanti. Nel dettaglio le istanze da 70 punti del dataset Circle arrivano addirittura ad un 26% in più rispetto all'ottimo di CPLEX, evidenziando come alcune particolari distribuzioni di nodi comportino un aumento della complessità considerevole che mette in difficoltà l'algoritmo realizzato per il progetto.

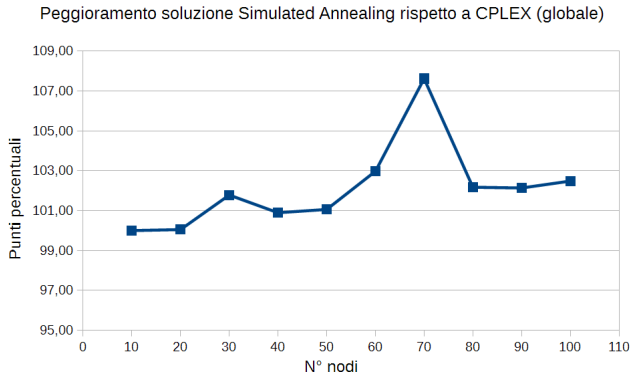

Figura 9: Grafico che riassume il peggioramento, in punti percentuali, dei risultati della Simulated Annealing rispetto al CPLEX

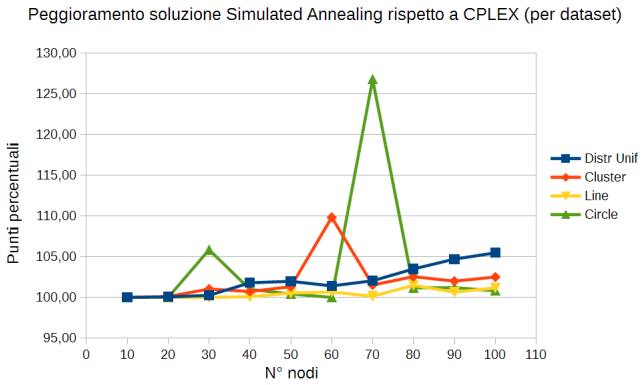

Figura 10: Grafico che riassume il peggioramento, in punti percentuali, dei risultati della Simulated Annealing rispetto al CPLEX

4.4. ***Altri test, risultati e considerazioni***

In questo paragrafo verranno commentati altri test condotti con diverse configurazioni rispetto a quelle precedentemente evidenziate nonché alcune considerazioni e risultati relativi ai dati fino ad ora esposti.

4.4.1. ***Limite di tempo CPLEX: cosa è emerso***

Durante i test eseguiti con CPLEX è stato impostato un limite di tempo entro cui la soluzione doveva essere calcolata, questo perché sono state individuate alcune istanze con cui il risolutore incontrava notevoli difficoltà e che anche dopo molte ore non produceva alcun tipo di risultato in output. Il comando con cui si è eseguito ciò è il seguente:

CPXsetdblparam(env, CPX_PARAM_TILIM, timelimit)

dove timelimit è stato impostato a 3600 secondi. Il numero di problemi che ha raggiunto tale upper bound sono solo 10 su 140, tutti collocati nella fascia di nodi 70 - 100, ed in questo caso si è preso come valore di riferimento per l'ottimo quello che era stato calcolato fino a quel momento. Come si può vedere quindi, ciò non ha costituito la "norma" per le varie istanze quanto piuttosto dei casi isolati che era comunque doveroso trattare adeguatamente.

4.4.2. ***Test con un maggior numero di passi***

Come evidenziato nel paragrafo 4.3, sono stati condotti test impostando un numero di passi dinamico molto più alto rispetto ai dati evidenziati in precedenza con l'obiettivo di verificare se effettivamente la soluzione trovata per i vari problemi fosse migliore. Nel dettaglio la formula utilizzata è stata la seguente:

n_passi = 2000000.0 * n / 5

I risultati conseguiti così facendo non sono stati affatto soddisfacenti in quanto, a fronte di un piccolissimo miglioramento della soluzione trovata si è ottenuto un aumento considerevole e molto più rapido dei tempi di esecuzione. Il fenomeno si può osservare nelle figure 11 e 12 dove è ben visibile come i tempi di esecuzione siano passati, per 100 nodi, da circa 11 secondi a più di 90, mentre il guadagno in termini di avvicinamento della soluzione all'ottimo è praticamente quasi nullo.

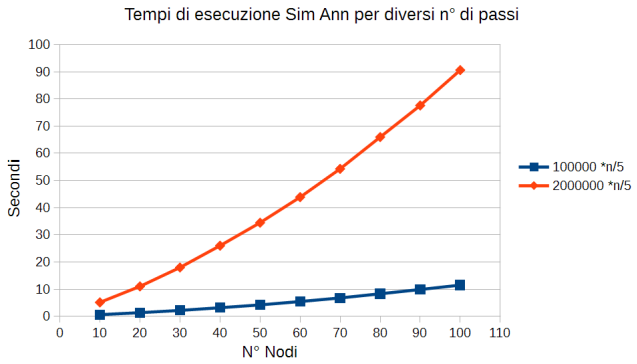

Figura 11: Grafico che riassume i tempi di esecuzione del Sim Ann utilizzando due diverse quantità dinamiche di passi per l'esecuzione

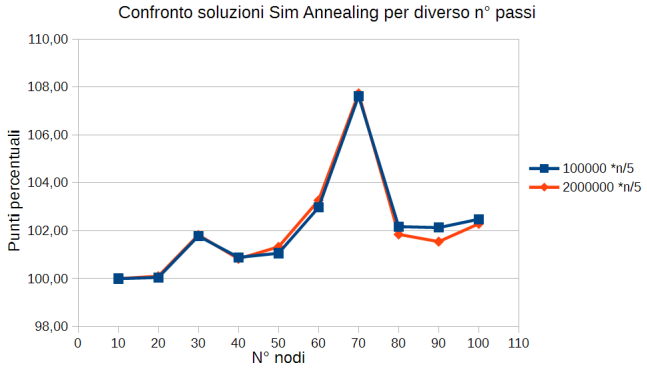

Figura 12: Grafico che riassume la distanza in punti percentuali delle soluzioni del Sim Ann utilizzando due diverse quantità dinamiche di passi per l'esecuzione

5. Conclusioni

Osservando i risultati ottenuti è emerso che il Simulated Annealing, rispetto al CPLEX, non riesce quasi mai a raggiungere l'ottimo ma con l'aiuto anche della Local Search si è riusciti a raffinare sufficientemente la soluzione per ottenere dei valori più che accettabili. Per migliorare i risultati ottenuti si è provato anche ad aumentare il numero di passi dinamici eseguiti ma ciò ha comportato solo un piccolo vantaggio a fronte di un aumento considerevole dei tempi di esecuzione. Ha quindi prevalso l'efficienza dell'algoritmo e si è optato per un numero di passi inferiore per i test di riferimento.

Paragonando ora i tempi di esecuzione delle due tecniche, abbiamo che dopo un iniziale vantaggio di CPLEX su Simulated Annealing, dai 40 punti in poi i valori si distanziano notevolmente arrivando, per 100 nodi, ad un'elevatissima differenza di 1428 secondi del primo contro gli 11 del secondo. Talvolta è stato addirittura necessario fermare l'esecuzione del risolutore CPLEX poiché anche dopo diverse ore non produceva alcun tipo di risultato, tuttavia i problemi particolarmente complessi in cui è stata necessaria questa manovra drastica erano pochi rispetto al totale.

A conti fatti quindi, i due algoritmi possiedono pregi che possiamo definire complementari: se si necessita di ottenere una soluzione ottima per i propri problemi e non si bada ai tempi di esecuzione allora CPLEX è ciò che è necessario utilizzare; viceversa, se è molto più importante ottenere risultati in tempi relativamente brevi Simulated Annealing è la scelta più ovvia da attuare per i nostri scopi.

5.1. ***Possibili miglioramenti del Simulated Annealing***

Per migliorare ulteriormente il Simulated Annealing sarebbe stato possibile implementare alcuni accorgimenti che avrebbero reso ancora più efficiente l'algoritmo; tuttavia, visti i già contenuti tempi di esecuzione rilevati, si è preferito optare per una maggior semplicità del codice così che risultasse più immediato da leggere ed, eventualmente, espandere. Gli accorgimenti sono i seguenti:

- Terminazione in caso di raggiungimento di un prefissato tempo limite
- Terminazione in caso di ottimalità della soluzione trovata
- Terminazione in seguito ad un certo numero di iterazioni senza miglioramento della soluzione