GANDTT GENERATOR

Come vi avevo anticipato, ho scritto un codice che rappresenti tramite GANDTT le soluzioni trovate dai vari script, ritengo siano molto più utili del previsto per capire bene i meccanismi di scelta ideali per eventuali solver ad-hoc. Potete vedere un esempio di GANDTT generato nel file FOTO1, ma per le prossime sezioni tanto ne vedrete altri.

BACKWARD EXACT ALGORITHM ATTEMPT

Come da accordi ho scritto un algoritmo che componesse, a partire da un ordine imposto per i pazienti, una schedule a partire dal fondo. L'algoritmo usato potete trovarlo nel file Backward_algorithm, tutto ciò che non è color fucsia è l'algoritmo, sebbene fosse promettente neanche questo risulta essere ottimale, in quanto di nuovo facendo una scelta che al momento sembra essere la migliore, capita che più avanti nella schedule questa scelta ci costringa a peggiorare l'Objective Function per un altro job (che andando in ordine di pazienti verrebbe prima, ma che andando in ordine di costruzione al contrario arriva dopo). Potete notare un esempio di soluzione fatta usando questo algoritmo in FOTO2, in cui il risveglio (verde) del paziente 3 viene messo nella posizione migliore al momento della scelta, ma che poi si traduce in una perdita di tempo quando poi andrà posizionato il risveglio del paziente 14. Ho tentato di modificare l'algoritmo per contare casi speciali (la parte di Backward_algorithm aggiunta in fucsia), quindi un nuovo algoritmo modificato, ma sebbene questo risolva il problema creatosi con il paziente 3, porta a creare altri problemi con i pazienti delle scelte successive, come potete notare in FOTO3. Ho anche provato a modificare ulteriormente l'algoritmo con ragionamenti del tipo "fai la scelta controintuitiva se succede che il prossimo paziente ti porta ad avere un miglioramento superiore a quella che è la perdita dovuta alla scelta controintuitiva", ma il problema è che ogni scelta che viene fatta per il posizionamento di un risveglio comporta una modifica a quella che è, per ogni paziente che si dovrà posizionare successivamente, la situazione in quanto a tempo disponibile, e la cosa si protrae per tutti i pazienti successivi, non solo per quello immediatamente dopo, motivo per cui in effetti ritengo che andare a scrivere un algoritmo che tenga in considerazione tutti i pazienti a venire (verifico che fare una scelta controintuitiva adesso mi permetta di riguadagnare tempo tra 5 pazienti), non sia altro che un andare a verificare tutte le possibili combinazioni di posizionamenti (cioè 2^n combinazioni).

GUROBI SOLVING OF PROBLEM 2

Per quanto riguarda il problema 2 invece (quello in cui si cerca anche l'ordine ottimale di pazienti) eravamo rimasti che avevo scritto un algoritmo che trattasse le varie fasi come una macchina, e quindi risolveva semplicemente imponendo il non-overlap delle operazioni, e che impiegava svariate ore per arrivare a soluzione ottimale con 15 pazienti irrealistici (specifico perché importante dopo).

Innanzi tutto, cosa meno rilevante, usando tempi realistici per le varie fasi (Anestesia<15min, Operazione<120min, Risveglio<20min), il vecchio solver è molto rapido, essendo un caso intuitivamente più semplice, e la soluzione è semplicemente che tutti i risvegli sono in AOP, come potete vedere in FOTO4.

Come già detto prima, se i processing time diventano generici come lunghezza, l'algoritmo non arriva a concludere la ricerca in meno di 6h (non l'ho mai lasciato proseguire oltre, non saprei a quanto arriva).

Se vi interessasse, potete vedere l'algoritmo in Gurobi2_old.

Nell'ultima settimana ho quindi cercato, mentre facevo il resto che ho già detto, un nuovo algoritmo per il problema 2, il risultato finale potete vederlo in Gurobi2_new, mentre in FOTO5 potete osservare un esempio di soluzione ottimale con 16 pazienti e processing time non realistici (quindi il caso peggiore dal punto di vista di complessità, molto più grande di quanto sia riuscito a testare con il vecchio algoritmo).

Come potete immaginare dal fatto che ve lo sto mostrando, questo nuovo algoritmo è estremamente più veloce del precedente, in particolare (per dare un'idea):

- **13 pazienti, tempi realistici, algoritmo vecchio:** circa 10secondi di runtime
- **18 pazienti, tempi realistici, algoritmo nuovo**: runtime istantaneo

- **15 pazienti, tempi irrealistici, algoritmo vecchio:** dopo 6h non ha ancora concluso la ricerca con un gap di svariati punti percentuale (non mi sono segnato quanti purtroppo)
- **14 pazienti, tempi irrealistici, algoritmo nuovo:** 2.6min runtime
- **15 pazienti, tempi irrealistici, algoritmo nuovo:** 4.5min runtime
- **16 pazienti, tempi irrealistici, algoritmo nuovo:** 25.6min runtime

Sebbene l'ultimo esempio possa sembrare molto lungo come runtime, c'è da dire che ha fatto esecuzione completa (quindi soluzione ottimale garantita, cioè nessuna necessità di rimettere mano alla schedule a posteriori come era necessario con il vecchio algoritmo) e che il problema era più grosso del caso peggiore che il vecchio algoritmo non era in grado di completare (banalmente, aggiungere un 16esimo paziente vuol dire moltiplicare le possibili combinazioni del caso a 15 pazienti per 32).