ABSTRACT

Il presente progetto riguarda la progettazione di un software in grado di risolvere istanze del problema "Il Commesso Viaggiatore" applicando differenti algoritmi risolutori. L'obiettivo di questo testo è quello di descrivere le tecniche utilizzate e di confrontare i risultati ottenuti in termini di efficienza e bontàà della soluzione prodotta. Verràà fornita una descrizione degli strumenti e l'ambiente di sviluppo utilizzati e saràà analizzato il codice di programmazione realizzato; non mancheranno paragrafi dedicati ad approfondire concetti teorici senza i quali la comprensione del codice potrebbe risultare meno chiara.

INTRODUZIONE

Questa capitolo introduttivo è dedicato alla storia, alle applicazioni e correnti sfide riguardanti uno dei più importanti problemi che la disciplina di Ricerca Operativa si trova ad affrontare, ossia il problema del commesso viaggiatore(Travelling Salesman Problem -TSP). Il nome deriva dalla sua più tipica rappresentazione: data una rete di cittàà, connesse tramite delle strade, si vuole trovare il percorso di minore distanza che un commesso viaggiatore deve seguire per visitare tutte le cittàà una ed una sola volta e ritornare alla cittàà di partenza. Per quanto detto, risulta naturale modellare il TSP come un grafo pesato i cui nodi modellizzano le cittàà relative al problema in questione mentre i possibili collegamenti tra le localitàà sono modellati con gli archi del grafo i cui pesi possono rappresentare,per esempio, la distanza esistente fra la coppia di nodi collegati dall'arco. Chiaramente è possibile assegnare i pesi in modo arbitrario secondo le nostre esigenze, ad esempio si potrebbe anche tenere conto dei tempi di percorrenza o eventuali pedaggi presenti nei singoli percorsi. Come è facile immaginare, il TSP può essere quindi utilizzato per una infinità di problemi pratici ma anche teorici:
Il problema del commesso viaggiatore riveste un ruolo notevole nell'ambito di problemi di logistica distributiva, detti anche di routing. Questi riguardano l'organizzazione di sistemi di distribuzione di beni e servizi. Esempi di problemi di questo genere sono la movimentazione di pezzi o semilavorati tra reparti di produzione, la raccolta e distribuzione di materiali, lo smistamento di merci da centri di produzione a di distribuzione. Sebbene le applicazioni nel contesto dei trasporti siano le più naturali per il TSP, la semplicitàà del modello ha portato a molte applicazioni interessanti in altre aree. Un esempio può essere la programmazione di una macchina per eseguire fori in un circuito. In questo caso i fori da forare sono le città e il costo del viaggio è il tempo necessario per spostare la testa del trapano da un foro all'altro. Il problema del commesso viaggiatore risulta essere NP-hard: questo significa che, al momento, non è noto in letteratura un algoritmo che lo risolva in tempo polinomiale. Poichéè esiste sempre una istanza per cui il tempo di risoluzione cresce esponenzialmente non è sempre possibile utilizzare algoritmi esatti per risolvere il TPS. Risulta quindi necessario fornire algoritmi euristici, in grado di risolvere in modo efficace istanze con un numero elevato di nodi in tempi ragionevoli.
Problemi matematici riconducibili al TSP furono trattati nell'Ottocento dal matematico irlandese Sir William Rowan Hamilton e dal matematico Britannico Thomas Penyngton. Nel 1857, a Dublino, Rowan Hamilton descrisse un gioco, detto Icosian game, a una riunione della British Association for the Advancement of Science. Il gioco consisteva nel trovare un percorso che toccasse tutti i vertici di un icosaedro, passando lungo gli spigoli, ma senza mai percorrere due volte lo stesso spigolo. L'icosaedro ha 12 vertici, 30 spigoli e 20 facce identiche a forma di triangolo equilatero. Il gioco, venduto alla ditta J. Jacques and Sons per 25 sterline, fu brevettato a Londra nel 1859, ma vendette pochissimo. Questo problema è un TSP nel quale gli archi che collegano vertici adiacenti, e quindi corrispondono a spigoli dell'icosaedro, sono consentiti e gli altri no (si può pensare che richiedano moltissimo tempo e quindi vadano sicuramente

scartati), per tale ragione si tratta di un caso molto particolare di TSP. La forma generale del TSP fu invece studiata solo negli anni Venti e Trenta del ventesimo secolo dal matematico ed economista Karl Menger. Tuttavia, per molto tempo non si ebbe altra idea che quella di generare e valutare tutte le soluzioni, il che mantenne il problema praticamente insolubile. Il numero totale dei differenti percorsi possibili attraverso le n cittàà è facile da calcolare: data una cittàà di partenza, ci sono a disposizione $(n-1)$ scelte per la seconda cittàà, $(n-2)$ per la terza e così via. Il totale delle possibili scelte tra le quali cercare il percorso migliore in termini di costo è dunque $(n-1)!$, ma dato che il problema ha simmetria, questo numero va diviso a metà. Insomma, date n cittàà, ci sono $\frac{(n-1)!}{2}$ percorsi che le collegano.
Solo nel 1954, George Dantzig, Ray Fulkerson e Selmer Johnson proposero un metodo più raffinato per risolvere il TSP su un campione di $n = 49$ cittàà: queste rappresentavano le capitali degli Stati Uniti e il costo del percorso era calcolato in base alle distanze stradali.
Nel 1962, Procter and Gamble bandì un concorso per 33 cittàà, nel 1977 fu bandito un concorso che collegasse le 120 principali cittàà della Germania Federale e la vittoria andò a Martin Grötschel oggi Presidente del Konrad-Zuse-Zentrum für Informarionstechnik Berlin(ZIB) e docente presso la Technische Universität Berlin(TUB).
Nel 1987 Padberg e Rinaldi riuscirono a completare il giro degli Stati Uniti attraverso 532 cittàà. Nello stesso periodo Groetschel e Holland trovarono il TSP ottimale per il giro del mondo che passava per 666 mete importanti. Nel 2001, Applegate, Bixby, Chvátal, and Cook trovarono la soluzione esatta a un problema di 15.112 cittàà tedesche, usando il metodo cutting plane, originariamente proposto nel 1954 da George Dantzig, Delbert Ray Fulkerson e Selmer Johnson. Il calcolo fu eseguito da una rete di 110 processori della Rice University e della Princeton University. Il tempo di elaborazione totale fu equivalente a 22,6 anni su un singolo processore Alpha a 500 MHz. Sempre Applegate, Bixby, Chvàtal, Cook, e Helsgaun trovarono nel Maggio del 2004 il percorso ottimale di 24,978 cittàà della Svezia. Nel marzo 2005, il TSP riguardante la visita di tutti i 33.810 punti in una scheda di circuito fu risolto usando CONCORDE: fu trovato un percorso di 66.048.945 unitàà, e provato che non poteva esisterne uno migliore. L'esecuzione richiese approssimativamente 15,7 anni CPU. Ai giorni nostri il risolutore Concorde per il problema del commesso viaggiatore è utilizzato per ottenere soluzioni ottime su tutte le 110 istanze della libreria TSPLIB; l' istanza con più nodi in assoluto ha 85,900 cittàà.

AMBIENTE DI SVILUPPO: Cplex, Visual Studio e C#

Il progetto è stato sviluppato in ambiente Windows, in particolare il sistema operativo scelto è Windows 10.
Cplex è la componente principale del progetto, prima di procedere è perciò necessario assicurarsi per potervi interagire dall'IDE che si desidera utilizzare. Il nostro gruppo ha deciso di adottare come linguaggio di programmazione il **C#**, successore del **C++** ed anch'esso orientato agli oggetti.
L'IDE più comune per chi desidera utilizzare tale linguaggio è senza dubbio **Visual Studio**, sviluppato dalla stessa Microsoft e distribuito gratuitamente. La versione utilizzata in questo progetto e dunque quella a cui si farà riferimento in questa guida è **Visual Studio Community 2017**. Una volta aperto l'installer reperibile al seguente indirizzo è sufficiente installare i pacchetti **Sviluppo per desktop .NET** e **Sviluppo di applicazioni desktop con C++** (vedremo in seguito perché sono necessari pacchetti C++).
Una volta terminata l'installazione assicuriamoci di aver installare nella nostra macchina una versione di cplex pari o superiore alla **12.7.0**, quella da noi utilizzata è più precisamente la **12.7.0**.
A questo punto non ci rimane altro che creare il nostro progetto all'interno di Visual Studio e di

collegarvi le apposite librerie di Cplex. Selezionare quindi dal menu a tendina **Visual C#** e quindi **App console (.NET Framework)**, forniamo il nome e percorso che desideriamo:

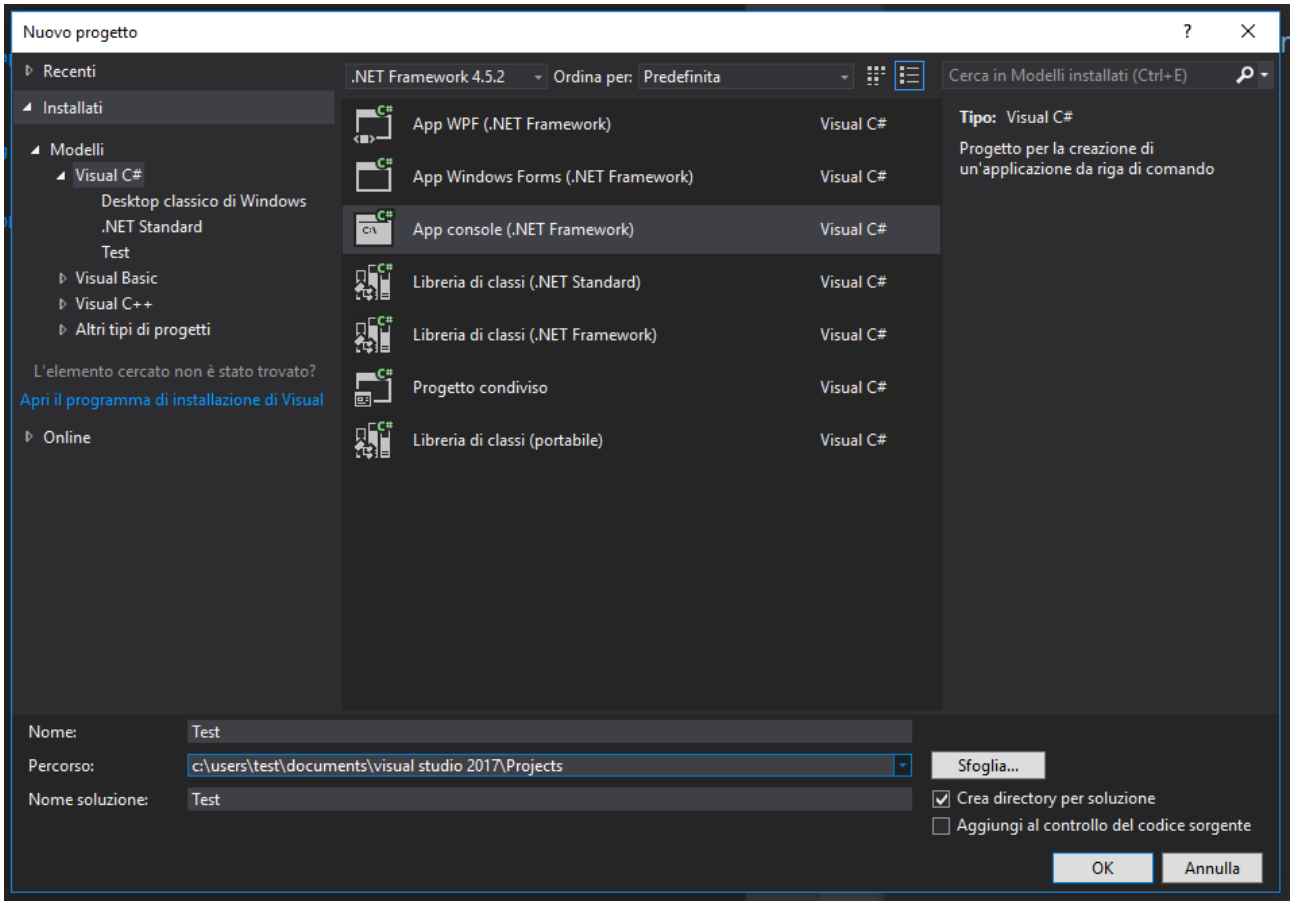

Figure 1: Creazione progetto C Sharp

Per connettere Cplex al nostro progetto sono necessari i seguenti passaggi:

- Selezionare la voce **Progetto** e quindi **Aggiungi riferimento...**
- Premere il pulsante **Sfoglia** e dopo essersi recati nella propria cartella di installazione di Cplex, in genere "C:\Program Files\IBM\ILOG\CPLEX_Studio1271\cplex" accedere alla sotto directory "\bin\x64_win64" e selezionare i file **ILOG.CPLEX.dll** e **ILOG.Concert.dll**. Fatto ciò possiamo chiudere la finestra per la gestione dei riferimenti appena aperta
- Selezionare la voce **Compilazione** e quindi **Gestione configurazione..**, nella nuova finestra inserire nella voce **Piattaforma** una nuova voce e selezionare **x64**. Per quanto riguarda la voce configurazione è indifferente selezionare la voce **Debug** oppure **Release**: come è facilmente intuibile nella prima modalità e differenza della seconda sarà possibile effettuare il classico debug con break points e gestione delle eccezioni a discapito di tempi di esecuzione leggermente più alti. Cogliamo quindi l'occasione per specificare che tutti i risultati proposti nel seguito di questo testo sono tutti stati ottenuti in modalità **Release** per il motivo indicato precedentemente.

A questo punto è possibile importare attraverso le direttive **using** sia **ILOG.CPLEX** che **ILOG.Concert**.

Creazione ed utilizzo DLL C/C++

In questa sezione viene spiegato come sia possibile utilizzare codice esterno, compilato anche in linguaggi differenti dal C#, sotto forma di **DLL**. In particolare nel nostro caso si è presentata la necessità di poter utilizzare codice scritto in linguaggio C appartenente al programma Concorde. La problematica principale con cui ci si scontra in questi casi è l'incompatibilità dei tipi, per noi questo è ancora più accentuato in quanto C al contrario di C# non è un linguaggio orientato agli oggetti e l'interfaccia con Cplex segue un differente approccio. La soluzione migliore è stata quindi quella di passare al codice C solamente le informazioni fornite in input dall'utente e cioè il nome del file contenente i dati ed il time limit. In altre parole il codice C# diventa solamente[1] una interfaccia per richiamare la DLL, la quale dovrà gestire completamente la lettura dell'input, la risoluzione e la visualizzazione dei risultati[2].
Entriamo ora nel dettaglio della procedura da seguire:

- Prima di tutto da Visual Studio creiamo un nuovo progetto selezionando la voce **Visual C++** e quindi **Progetto Win32**. Nel caso in cui questa opzioni siano assenti significa che durante l'installazione di Visual Studio non sono stati selezionati i pacchetti C++, per maggiori dettagli seguire la procedura indicata nella sezione apposita.
- Nelle schermate successive è necessario selezionare l'opzione **DLL** come tipo di applicazione e **Progetto vuoto** come opzione aggiuntiva.
- A questo punto dovrebbe essersi creato il nostro nuovo e vuoto progetto, dal menu **Esplora soluzioni** nella cartella **File di origine** premiamo il tasto destro ed aggiungiamo un nuovo elemento. Selezioniamo dal menu **File di C++ (.cpp)**, assegniamo il nome che preferiamo e premiamo il tasto **Aggiungi**.
- A questo punto il file appena creato verrà compilato come codice **C++** ma Concorde utilizza **C** perciò per prima cosa dobbiamo inserire

```
1 extern "C"
  {
3 }
```

  In questo modo tutto il codice al suo interno viene compilato come **C**.
- Definiamo ora l'entry point per la nostra DLL che si tratta di un semplice metodo con prefisso **__declspec(dllexport)**. Nel nostro caso si è deciso di utilizzare il metodo entry point come semplice interfaccia per la chiamata di un altro metodo che procederà alla reale risoluzione del problema:

```
1 __declspec(dllexport) int Concorde(char *fileName, int timeLimit)
  {
3 return exemain(strtok(fileName, "\0"), timeLimit);
  }
```

- In modo analogo a come abbiamo appena creato questo file selezionando invece l'opzione per aggiungere file esistenti, importiamo tutti i file con estensione **.C** di cui necessitiamo da Concorde (maggiori dettagli riguardo quelli da noi utilizzati possono essere trovati nella sezione apposita).

[1] In realtà mantiene anche un cronometro per il tempo di risoluzione
[2] Tutte le stampe effettuate nella DLL vengono automaticamente reindirizzate nella finestra Console di default utilizzata dal C#

- Settiamo ora le proprietà del progetto in modo tale che sia possibile utilizzare Cplex, diverse guide sono già disponibili e la procedura non viene qui riportata. In aggiunta è necessario selezionare nel sottomenù **C/C++** → **Generale** → **Directory di inclusione aggiuntive** la cartella dove sono presenti i file **.h** di Concorde.

- A questo punto non rimane altro che completare il nostro progetto inserendo i vari metodi necessari, maggiori dettagli nella sezione apposita. Fatto ciò siamo pronti a creare la DLL, dal menu **Compilazione** selezioniamo la voce **Compila soluzione** (assicuriamoci che nella barra degli strumenti sia selezionata la modalità release a 64 bit, in realtà è importante si utilizzino gli stessi bit che si adottano nel progetto originale). Terminata la compilazione, la DLL si può trovare all'interno della cartella del progetto nella sottocartella **/x64/Release**

- Posizioniamo la DLL all'interno della directory del nostro progetto C# in particolare dentro **/bin/x64/Release** e **/bin/x64/Debug** per essere utilizzata nelle due modalità.

- Apriamo infine la classe del nostro progetto nella quale vogliamo utilizzare la DLL ed utilizziamo questo codice:

```
[DllImport("ConcordeDLL.dll")]
public static extern int Concorde(StringBuilder fileName, int timeLimit);
```

  Successivamente potremmo utilizzare il metodo esterno importato come se fosse stato definito all'interno della classe, nel nostro caso:

```
Concorde(new StringBuilder(instance.InputFile), (int)instance.TimeLimit);
```

  Da notare che la classe **StringBuilder** permette la conversione automatica del tipo **String** di C# a **char*** di C.