Università degli Studi di Bari Aldo Moro
CdLM in Sicurezza Informatica

Sistema di consegna medicinali tramite drone nel Gargano

CORSO: METODI FORMALI PER LA SICUREZZA

DOCENTE: PROF.SSA FRANCESCA ALESSANDRA LISI

STUDENTE: GIACOMO PAGLIARA

MATRICOLA: 799628

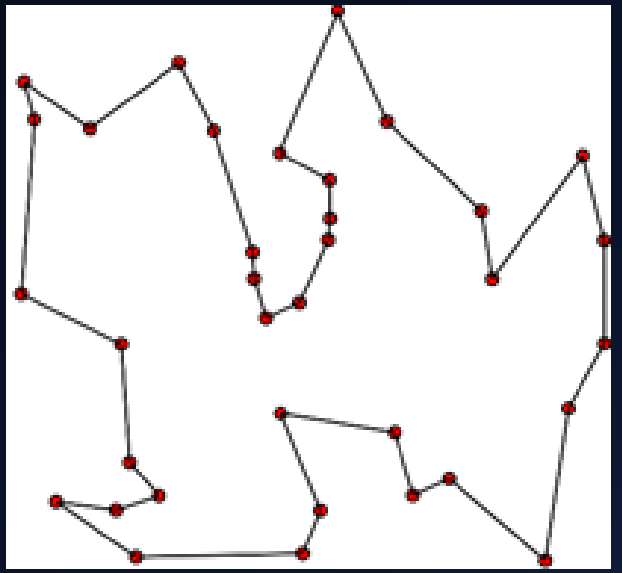
Introduzione al problema del commesso viaggiatore TSP

- Il **TSP (Travelling Salesman Problem)** è un problema di ottimizzazione in cui un viaggiatore deve visitare un insieme di città **esattamente una volta** e tornare al punto di partenza, minimizzando la distanza percorsa.

- **Origine storica:** Primo riferimento nel **1832** in un manuale tedesco. Studio formale con il **"Icosian Game"** di Hamilton (introduzione dei cicli hamiltoniani).

Il problema e la sua complessità

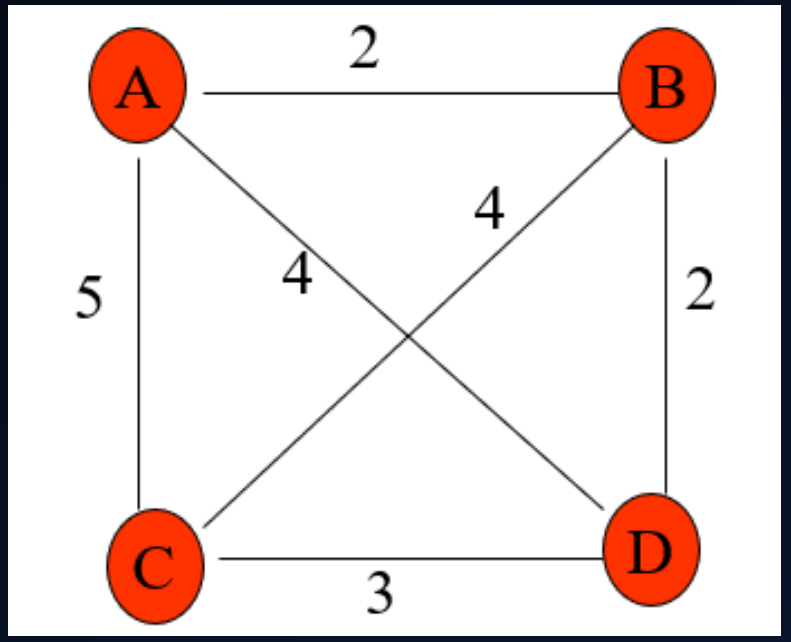
- Questione fondamentale: «**Un commesso viaggiatore deve guidare fino a N città sulla mappa e poi tornare a casa. Vuole spendere il meno possibile in benzina. Come troviamo il percorso più breve da seguire?**»
- Sfida computazionale:
  Crescita esponenziale (N!)
  Esempio: 10 punti → 3,6 milioni di percorsi.
- Il TSP è un problema **NP-hard** → Non esiste un algoritmo efficiente che garantisca sempre la soluzione ottimale in tempi ragionevoli.
- Anche i **supercomputer** non riescono a risolverlo esattamente per grandi N



Formalizzazione Matematica

Grafo G = (V, A)

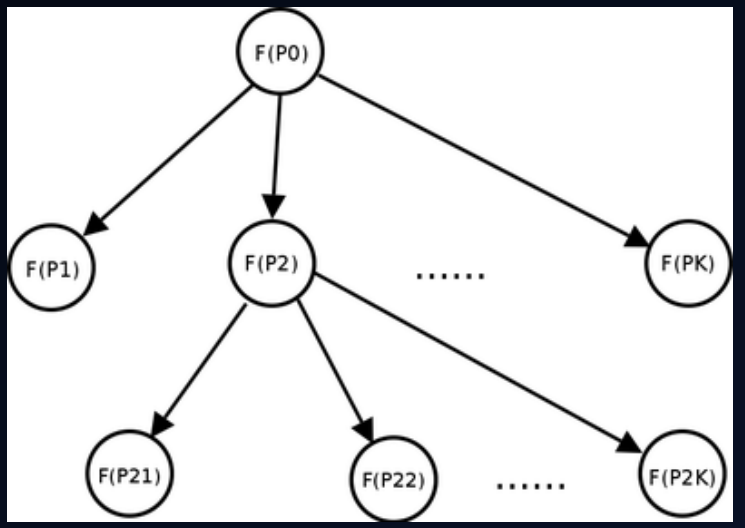
- V: insieme degli n nodi (città);
- A: insieme degli archi (strade);
- TSP simmetrico vs asimmetrico;
- Funzione obiettivo: minimizzazione del costo cammino.



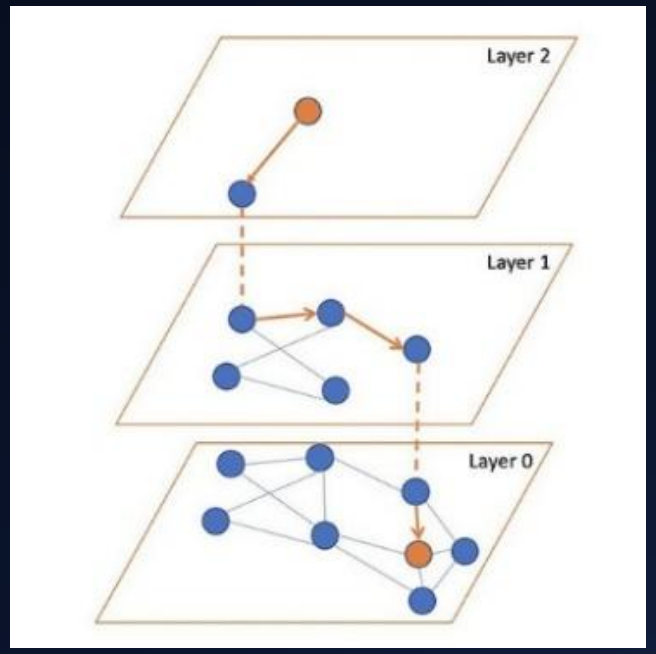
Metodi di Risoluzione del TSP

Due categorie di approcci:

- **Algoritmi esatti:** la soluzione viene ricercata in maniera esaustiva, ma molto spesso non è fattibile a causa del lungo tempo di calcolo richiesto.
- Algoritmi euristici: permettono di trovare risposte approssimative in meno tempo, esistono vari sottotipi:



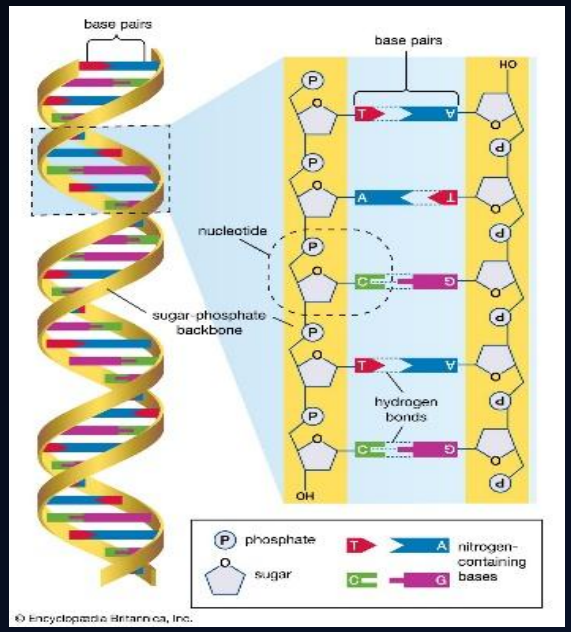
- **Forza bruta**: Prova tutti i percorsi possibili → impraticabile per grandi N.
- **Branch and Bound**: Esplora solo le soluzioni promettenti per ridurre il tempo di calcolo. Si concentra sull'ottimizzazione.
- ***Metodo del vicino più prossimo***: parte da un nodo casuale, aggiunge il nodo più vicino e ripete il processo fino a visitare tutte le destinazioni. È semplice ma non sempre fornisce la soluzione ottimale

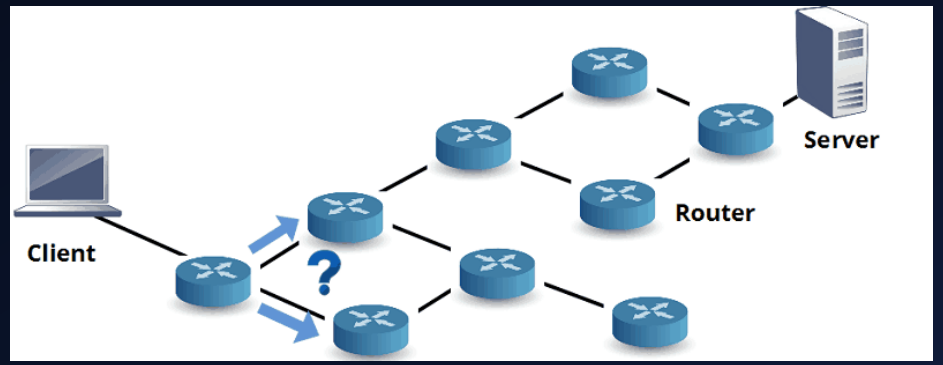


Applicazioni Reali del TSP

Si applica in molti ambiti reali:

- 🚚 **Logistica e trasporti**: Ottimizzazione delle consegne e riduzione del carburante.
- 🤖 **Robotica**: Movimenti efficienti per bracci robotici e droni.
- 🌐 **Routing di rete**: Ottimizzazione del flusso di dati in Internet.
- 🧬 **Biologia**: Sequenziamento del DNA per ridurre il tempo di analisi.
- 🏭 **Pianificazione industriale**: Ottimizzazione dei cicli produttivi nelle fabbriche.





Codifica ASP del TSP

- Analisi della codifica ASP (Answer Set Programming) del TSP
- Contesto: consegna farmaci mediante drone
- Partenza: Ospedale di San Giovanni Rotondo
- Destinazioni: località del Gargano e Isole Tremiti

- Motivazioni della scelta:
  - Necessità di consegna rapida in aree complesse
  - Presenza di zone costiere e isole
  - Varietà di terreni e altitudini significative
  - Ottimizzazione del trasporto medico

Struttura Generale e Nodi

- Punto di Partenza:
  - San Giovanni Rotondo (start)
  - Altitudine: 500m
  - Sede dell'ospedale

- Località Costiere:
  - Vieste (nodo 1, 40m)
  - Peschici (nodo 2, 90m)
  - Rodi Garganico (nodo 3, 40m)

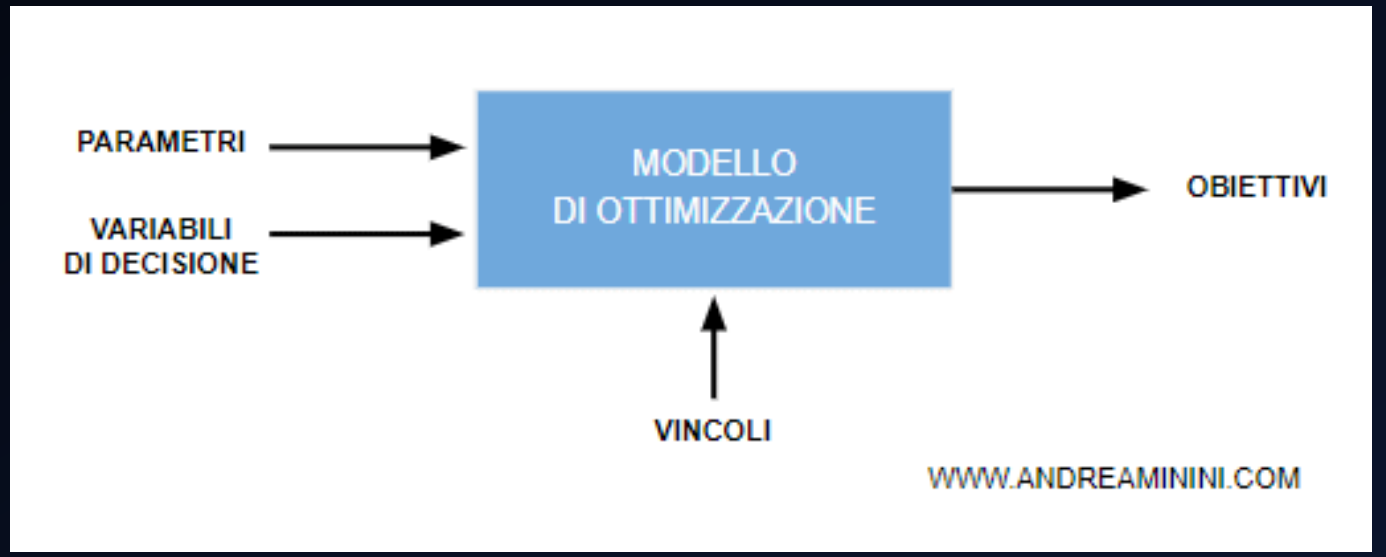
- Isole Tremiti:
  - Isola di San Domino (nodo 4, 70m)
  - Isola di San Nicola (nodo 5, 70m)

- Entroterra:
  - Monte Sant'Angelo (nodo 6, 700m)

Sfide Operative

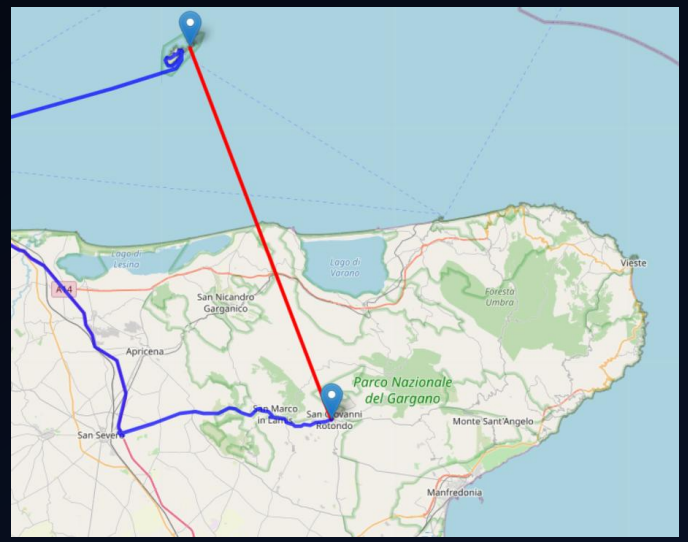
- Vincoli Operativi:
  - Gestione consumo energetico sui tratti marittimi
  - Impatto dei dislivelli delle altitudini sulle prestazioni
  - Ottimizzazione del percorso

- Assunzioni:
  - Carico medicinali entro limiti consentiti



Funzione di Costo

- Distanza Fisica:
  - Distanza in linea d'aria tra località (km)
  - Calcolata usando coordinate geografiche reali
  - Utilizzo di freemaptools.com per calcoli precisi
- Fattore Terreno:
  - Terreno terrestre: valore 1
  - Terreno marittimo: valore 1.5
    - Maggior consumo energetico
    - Necessità di stabilizzazione
    - Assenza punti atterraggio emergenza
- Fattore Altitudine:
  - Basato su altitudini reali
  - Impatto su consumo energetico
  - Gestione dislivelli significativi



Codifica ASP: Struttura e Predicati

Generazione dei cicli:

```
% Generate
{ cycle(X,Y) : edge(X,Y) } = 1 :- node(X).
{ cycle(X,Y) : edge(X,Y) } = 1 :- node(Y).
```

- Choice rules che generano il ciclo selezionando esattamente:
- un arco uscente per ogni nodo X
- un arco entrante per ogni nodo Y
- cycle(X,Y): predicato che rappresenta gli archi selezionati nella soluzione

Definizione raggiungibilità:

```
% Define
reached(Y) :- cycle(start,Y).
reached(Y) :- cycle(X,Y), reached(X).
```

Regole ricorsive che implementano la chiusura transitiva:

- nodi direttamente raggiungibili da start
- propagazione lungo il percorso scelto
- reached(Y): predicato che indica i nodi raggiungibili nel ciclo

Vincoli di validità:

```
% Test
:- node(Y), not reached(Y).
:- not reached(start).          % assicura il ritorno a start
```

- Garantisce visita di tutti i nodi
- Assicura ritorno al punto di partenza

Codifica ASP: Struttura e Predicati

Definizione Nodi

```
% Nodes
node(start).          % San Giovanni Rotondo (punto di partenza)
node(1..6).          % Altri nodi da visitare
```

Definizione Archi:

```
% Edges
edge(start, (1;2;3;4;5;6)).   % Da SGR posso andare ovunque - San Giovanni Rotondo
edge(1, (2;3;4;5;6)).         % Da Vieste posso andare agli altri (non SGR)
edge(2, (1;3;4;5;6)).         % Da Peschici posso andare agli altri (non SGR)
edge(3, (1;2;4;5;6)).         % Da Rodi posso andare agli altri (non SGR)
edge(4, (1;2;3;5;6)).         % Da San Domino posso andare agli altri (non SGR)
edge(5, (1;2;3;4;6)).         % Da San Nicola posso andare agli altri (non SGR)
edge(6, (1;2;3;4;5)).         % Da Monte Sant'Angelo posso andare agli altri (non SGR)
edge((1;2;3;4;5;6), start).   % Da tutti posso tornare a SGR
```

Distanze:

```
% Distance (in km, distanze reali in linea d'aria)
distance(start,1,55). distance(1,start,55).    % SGR-Vieste
distance(start,2,40). distance(2,start,40).    % SGR-Peschici
distance(start,3,30). distance(3,start,30).    % SGR-Rodi
distance(start,4,50). distance(4,start,50).    % SGR-San Domino
distance(start,5,53). distance(5,start,53).    % SGR-San Nicola
distance(start,6,25). distance(6,start,25).    % SGR-Monte Sant'Angelo
```

Sono state inserite solo alcune distanze in questo esempio

Codifica ASP: Struttura e Predicati

Fattore Terreno

```
% Terrain factor (1=terra, 15=mare che rappresenta 1.5)
terrain(start,1,1). terrain(1,start,1).   % terra (SGR-Vieste)
terrain(start,2,1). terrain(2,start,1).   % terra (SGR-Peschici)
terrain(start,3,1). terrain(3,start,1).   % terra (SGR-Rodi)
terrain(start,4,15). terrain(4,start,15). % mare (SGR-San Domino)
terrain(start,5,15). terrain(5,start,15). % mare (SGR-San Nicola)
terrain(start,6,1). terrain(6,start,1).   % terra (SGR-Monte Sant'Angelo)
```

- Si definisce il fattore *terreno* per ogni arco. Infatti, sono due i possibili valori per T: 1 per tratti terrestri (1.0) e 15 per tratti marittimi (che rappresenta 1.5 dopo la normalizzazione, infatti nella formula finale questo fattore verrà diviso per 10).

Codifica ASP: Struttura e Predicati

Fattore Altitudine

```
% Altitude
altitude(start,500). % San Giovanni Rotondo
altitude(1,40).      % Vieste
altitude(2,90).      % Peschici
altitude(3,40).      % Rodi Garganico
altitude(4,70).      % San Domino
altitude(5,70).      % San Nicola
altitude(6,700).     % Monte Sant'Angelo
```

Calcolo Costo Altitudine

```
% Altitude cost
alt_cost(X,Y,C) :- altitude(X,AX), altitude(Y,AY),
                   C = |AX-AY|/100.
```

- X: nodo di partenza;
- Y: nodo di arrivo;
- AX, AY: altitudini rispettive;
- C: costo normalizzato (differenza/100)
- Si ottiene un valore C che rappresenta l'Impatto della variazioni di altitudine sul costo Totale:
  - Esempio: 500m di dislivello = 5% di costo extra

Codifica ASP: Struttura e Predicati

Calcolo Costi e Ottimizzazione

```
% Final cost = (distance * terrain_factor/10) * (1 + altitude_cost)
cost(X,Y,C) :- distance(X,Y,D), terrain(X,Y,T), alt_cost(X,Y,A),
               C = (D * T / 10) * (1 + A).
```

Questa regola "cost(X,Y,C)" definisce il costo C per il tratto da X a Y dove: X è il nodo di partenza, Y nodo di arrivo e C il costo totale calcolato.

- Componenti:
  - **Prima parte:** (D * T / 10)
    - D: distanza in km
    - T: fattore terreno (1 o 15)
    - Divisione per 10 per normalizzare e quindi ottenere il costo base del tratto.
  - **Seconda parte:** (1 + A) aggiunge un costo percentuale basato sul dislivello;
  - «1» è necessario per garantire che la funzione di costo non sia mai zero e per mantenere il costo originale quando c'è dislivello l'addizione infatti permette di interpretare A come percentuale aggiuntiva trasformando il dislivello in un incremento proporzionale e mantenendo una relazione lineare tra dislivello e aumento di costo
    - A: differenza altitudine/100

Esempi pratici

Nessun dislivello: A = 0 (stessa altitudine), (1 + 0) = 1, nessun costo aggiuntivo.

Piccolo dislivello: A = 0.5 (50m di differenza), (1 + 0.5) = 1.5, aumento del 50% del costo.

Grande dislivello: A = 4.6 (460m di differenza), (1 + 4.6) = 5.6 aumento del 460% del costo.

Codifica ASP: Struttura e Predicati

…ritornando alla Funzione di Costo

```
% Final cost = (distance * terrain_factor/10) * (1 + altitude_cost)
cost(X,Y,C) :- distance(X,Y,D), terrain(X,Y,T), alt_cost(X,Y,A),
               C = (D * T / 10) * (1 + A).
```

- Quindi, il costo finale C per l'arco "(X,Y)" è dato dal prodotto di queste due parti. La formula tiene conto sia della lunghezza dell'arco, modificata dal tipo di terreno, sia delle variazioni di altitudine che influiscono sul consumo del drone.

- ***Esempio pratico:*** *per il tratto SGR → Vieste: D = 55 km (distanza), T = 1 (terreno terrestre), A = 4.6 (\|500-40\|/100). C = (55 * 1/10) * (1 + 4.6) = 5.5 * 5.6 = 30.8.*

Codifica ASP: Struttura e Predicati

Calcolo Costi e Ottimizzazione

```
% Optimize
#minimize { C,X,Y : cycle(X,Y), cost(X,Y,C) }.
```

- Mentre questa direttiva di minimizzazione comunica al solver ASP di minimizzare la somma dei costi C di tutti gli archi selezionati nel percorso definiti dal predicato "cycle/2".
- Il solver cercherà il ciclo Hamiltoniano che minimizza il costo totale calcolato con la funzione di costo.

```
% Display
#show cycle/2.
```

- Questa direttiva indica al solver di mostrare gli atomi relativi al predicato "cycle/2" nell'output finale. In questo modo, l'output si focalizzerà sul percorso cioè, sugli archi selezionati che fanno parte del ciclo.

Risultati

```
clingo version 5.7.0
Reading from stdin
Solving...
Answer: 1
cycle(start,3) cycle(6,5) cycle(5,2) cycle(4,6) cycle(3,4) cycle(2,1) cycle(1,start)
Optimization: 1679
Answer: 2
cycle(start,3) cycle(6,start) cycle(5,2) cycle(4,5) cycle(3,4) cycle(2,1) cycle(1,6)
Optimization: 215
Answer: 3
cycle(start,3) cycle(6,start) cycle(5,4) cycle(4,2) cycle(3,5) cycle(2,1) cycle(1,6)
Optimization: 213
OPTIMUM FOUND

Models       : 3
  Optimum    : yes
Optimization : 213
Calls        : 1
Time         : 0.047s (Solving: 0.01s 1st Model: 0.00s Unsat: 0.00s)
CPU Time     : 0.000s
```

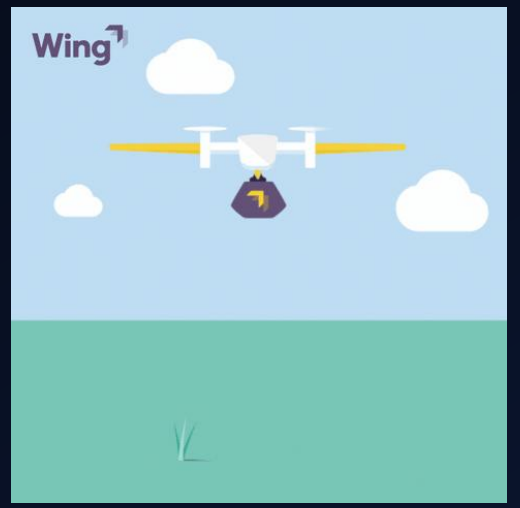
Analizzando i risultati ottenuti il percorso hamiltoniano ottimizzato è il seguente:

San Giovanni Rotondo(start) – Rodi Garganico(3) – San Nicola(5) – San Domino(4) – Peschici(2) – Vieste(1) – Monte Sant'Angelo(6) – start.


San Nicola
di Tremiti
Peschici
Rodi
Garganico
Torre Mileto
Capoiale
Lèsina
Lago di Varano
Vico del
Gargano
Spiaggia
Scialmarino
Vieste
SS89
SS693
SS89
Carpino
San Nicandro
Garganico
Cagnano
Varano
SS89
Parco
Nazionale
del Gargano
Pugnochiuso
SS89
Baia delle
Zagare
SS272
San Giovanni
Rotondo
SS272
Monte
Sant'Angelo
Mattinata

Modellazione in LTL/CTL

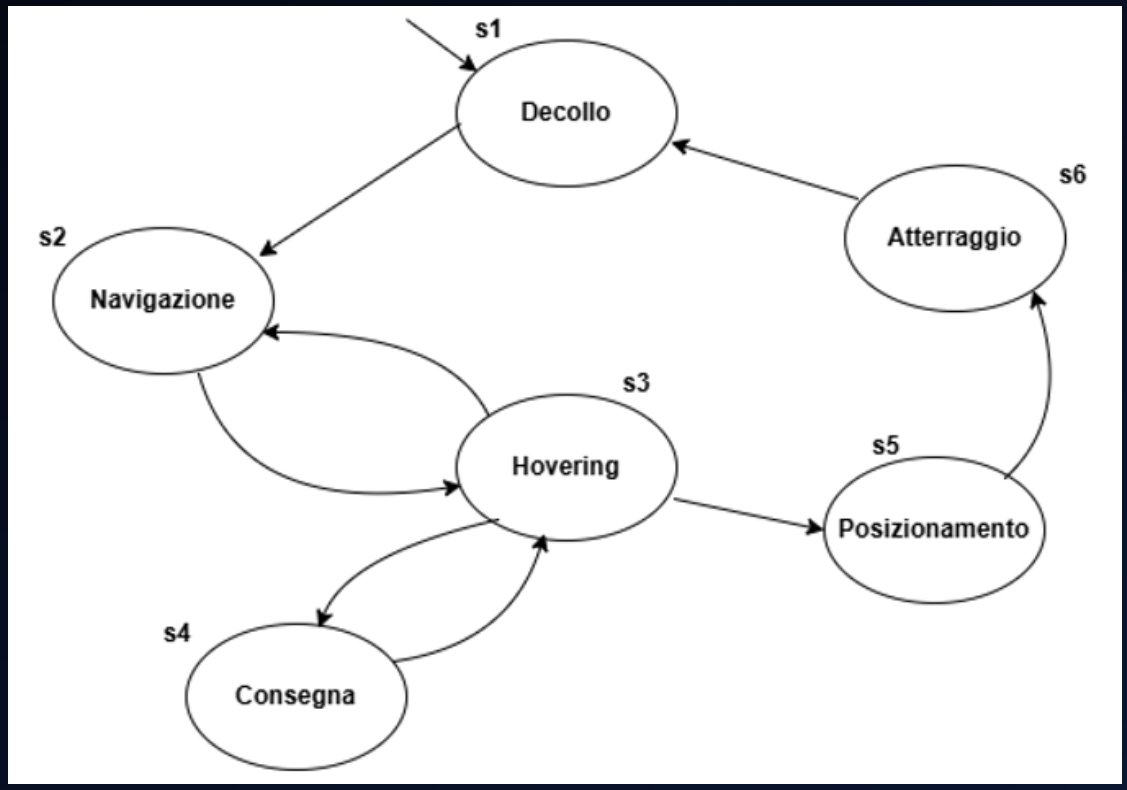
- Stati definiti nel sistema:
  - **Decollo**: stato iniziale, inizio del volo
  - **Navigazione**: segue il percorso pianificato
  - **Hovering**: stazionamento stabile per decisioni
  - **Consegna**: rilascio del carico medico
  - **Posizionamento**: allineamento preciso per fase finale
  - **Atterraggio**: completamento fase del percorso
- Note:
- Consegna: quota costante di sicurezza, verricello per rilascio
- Posizionamento: gestione dislivelli e riduzione graduale quota



Struttura di Kripke

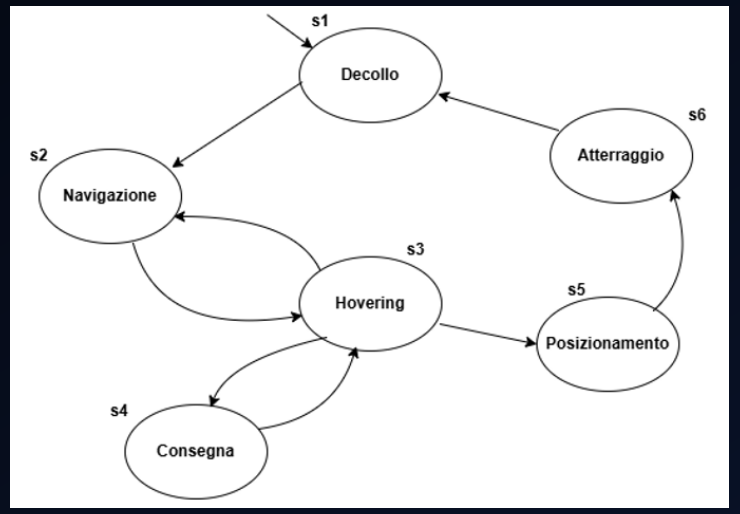
Definizione formale: M = (S, I, R, L)

- S = {s1, s2, s3, s4, s5, s6}
- I = {s1}
- R = {(s1,s2), (s2,s3), (s3,s2), (s3,s4), (s3,s5), (s5,s6), (s6,s1), (s4,s3)}
- L(s1) = {Decollo}
- L(s2) = {Navigazione}
- L(s3) = {Hovering}
- L(s4) = {Consegna}
- L(s5) = {Posizionamento}
- L(s6) = {Atterraggio}



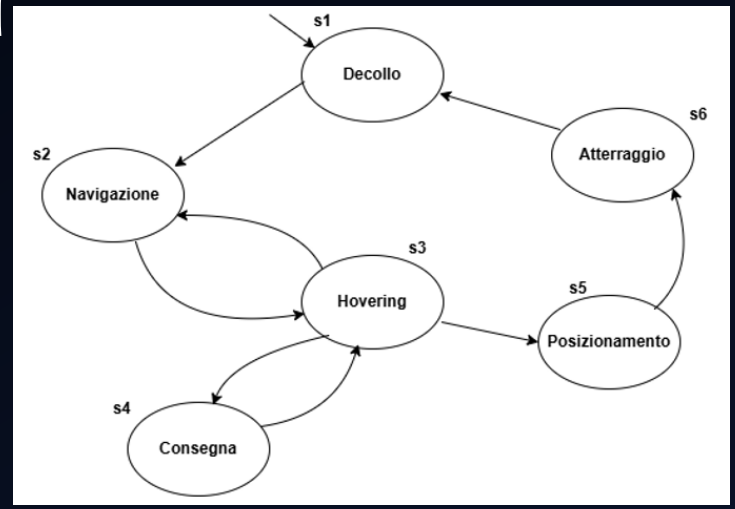
Proprietà LTL

- **G (decollo -> X navigazione)** : Dopo il decollo, il prossimo stato deve essere navigazione.
- **G (navigazione -> X hovering)** : Dopo la navigazione, il prossimo stato deve essere hovering.
- **G (hovering -> X (navigazione V consegna V posizionamento))** : Da hovering, il prossimo stato può essere navigazione, consegna o posizionamento.
- **G (consegna -> X hovering)** : Dopo la consegna si torna sempre in Hovering.
- **G (posizionamento -> X atterraggio)** : Dopo il posizionamento deve seguire l'atterraggio
- **G (hovering -> (X consegna -> X X hovering))** : Se da hovering si va in consegna, dopo si tornerà in hovering.
- **G (atterraggio -> F decollo)** : Dopo l'atterraggio, prima o poi si tornerà in decollo.



Proprietà CTL

- **AG (EF decollo)** : Da ogni stato è sempre possibile raggiungere, ad un certo punto, il decollo.
- **AG (decollo -> EF consegna)** : Dal decollo è sempre possibile, in futuro, raggiungere la consegna.
- **AG ¬(navigazione ∧ hovering)** : Non si può essere contemporaneamente in navigazione e hovering.
- **AG (navigazione -> EF hovering)** : Dalla navigazione si può sempre raggiungere, a un certo punto, hovering.
- **AG (hovering -> EF (posizionamento ∧ EX atterraggio))** : Da hovering esiste un percorso che lo porta, prima a entrare in posizionamento e poi raggiunge l'atterraggio.
- **AG (hovering -> E (hovering U (consegna ∨ posizionamento)))** : Da hovering esiste un percorso dove si rimane in hovering fino a raggiungere consegna o posizionamento.



Codifica NuSMV

Modulo main definisce il sistema che ho modellato:

- Ho definito una variabile «state» che può assumere 6 valori (s1...s6), rappresentando gli stati del sistema
- Per ogni stato, abbiamo anche una variabile booleana corrispondente
- Questa doppia rappresentazione (state + booleane) ci permette di:
  - Tracciare facilmente lo stato corrente del sistema
  - Verificare facilmente le proprietà temporali
  - Avere una rappresentazione più chiara delle condizioni del drone

```
MODULE main
VAR
    state : {s1, s2, s3, s4, s5, s6};
    decollo : boolean;
    navigazione : boolean;
    hovering : boolean;
    consegna : boolean;
    posizionamento : boolean;
    atterraggio : boolean;
```

Codifica NuSMV

Assegnazioni iniziali:

- Il sistema parte dallo stato s1 (decollo)
- Solo la variabile decollo è inizialmente TRUE
  - Il drone deve necessariamente iniziare dal decollo
  - Garantiamo uno stato iniziale ben definito
  - Evitiamo stati ambigui o inconsistenti.

Transizioni di stato:

- Definisce come il sistema evolve tra gli stati
- Punto chiave: da s3 (hovering) abbiamo tre possibili transizioni
- Da hovering può riprendere la navigazione, effettuare una consegna o prepararsi all'atterraggio
- Questa struttura riflette la reale operatività di un drone

```
ASSIGN
    init(state) := s1;
    init(decollo) := TRUE;
    init(navigazione) := FALSE;
    init(hovering) := FALSE;
    init(consegna) := FALSE;
    init(posizionamento) := FALSE;
    init(atterraggio) := FALSE;

    next(state) :=
        case
            state = s1 : s2;  -- da decollo a navigazione
            state = s2 : s3;  -- da navigazione a hovering
            state = s3 : {s2, s4, s5};  -- da hovering a navigazione, consegna o posizionamento
            state = s4 : s3;  -- da consegna a hovering
            state = s5 : s6;  -- da posizionamento a atterraggio
            state = s6 : s1;  -- da atterraggio a decollo
        esac;
```

Codifica NuSMV

Aggiornamento delle variabili booleane:

- Per ogni proprietà (decollo, navigazione, ecc.) ho definito che essa diventi TRUE nel passo successivo, se il prossimo stato corrisponde a quello associato
- Mantiene sincronizzazione tra stato e variabili booleane
  - Semplifica la verifica delle proprietà
  - Rende il modello più leggibile e verificabile

```
next(decollo) :=
    case
        next(state) = s1 : TRUE;
        TRUE : FALSE;
    esac;

next(navigazione) :=
    case
        next(state) = s2 : TRUE;
        TRUE : FALSE;
    esac;
```

Verifica delle proprietà:

- Le proprietà sono specificate usando le keyword LTLSPEC e CTLSPEC che poi vengono verificate automaticamente dal model checker.

```
-- Proprietà CTL
CTLSPEC AG(EF decollo)
```

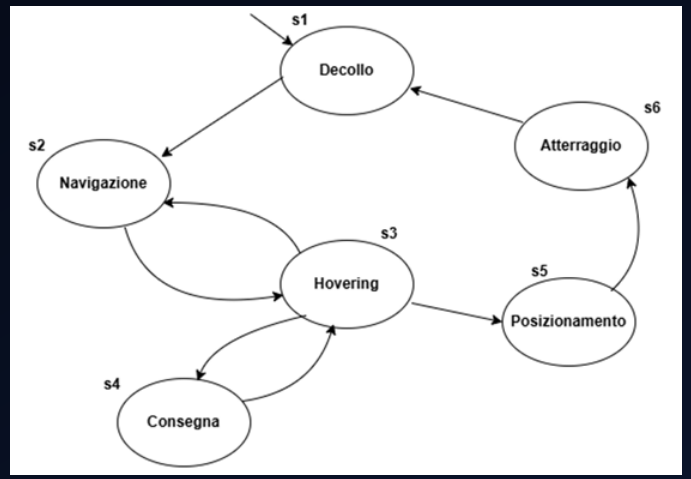
- Il sistema conferma se ogni proprietà è soddisfatta o fornisce un controesempio.

```
-- Proprietà LTL
LTLSPEC G(decollo -> X navigazione)
```

Casi di non soddisfacimento

❌ **Violazione della Proprietà LTL: G (decollo -> X navigazione)** ❌

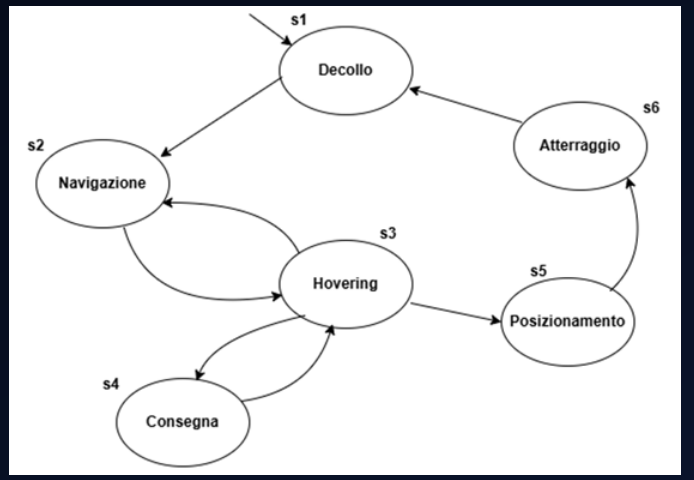
- **Passo 0:**
  - Il sistema parte da **s1** (Decollo è TRUE)
- **Passo 1 (controesempio):**
  - Il sistema sceglie la transizione **s1** → **s1** (rimane in Decollo) anziché passare a **s2** (Navigazione)
- **Implicazione:**
  - La proprietà **G (decollo -> X navigazione)** viene violata, poiché, partendo da s1, il prossimo stato non è Navigazione come richiesto
  - La proprietà globale fallisce a causa del mancato rispetto della transizione prevista



Casi di non soddisfacimento

❌ **Violazione Proprietà CTL: AG (hovering -> E (hovering U (consegna V posizionamento)))** ❌

- **Scenario:**
  - Da **s1 (Decollo)** → **s2 (Navigazione)** → **s3 (Hovering)**
- **Controesempio:**
  - Da **s3 (Hovering)** il sistema va direttamente a **s6 (Atterraggio)**, senza passare per **s4 (Consegna)** o **s5 (Posizionamento)**
  - il percorso ipotetico seguito è **s1 (decollo)** → **s2 (navigazione)** → **s3 (hovering)** → **s6 (atterraggio) …**
  - La proprietà **AG (hovering -> E (hovering U (consegna V posizionamento)))** viene violata



Risultati

Prompt dei comandi

```
C:\Users\giaco\Desktop\Sicurezza Informatica\2_ANNO\Metodi Formali per la Sicurezza\NuSMV-2.6.0-win64\bin>nusmv drone.smv
*** This is NuSMV 2.6.0 (compiled on Wed Oct 14 15:37:51 2015)
*** Enabled addons are: compass
*** For more information on NuSMV see <http://nusmv.fbk.eu>
*** or email to <nusmv-users@list.fbk.eu>.
*** Please report bugs to <Please report bugs to <nusmv-users@fbk.eu>>

*** Copyright (c) 2010-2014, Fondazione Bruno Kessler

*** This version of NuSMV is linked to the CUDD library version 2.4.1
*** Copyright (c) 1995-2004, Regents of the University of Colorado

*** This version of NuSMV is linked to the MiniSat SAT solver.
*** See http://minisat.se/MiniSat.html
*** Copyright (c) 2003-2006, Niklas Een, Niklas Sorensson
*** Copyright (c) 2007-2010, Niklas Sorensson

-- specification AG (EF decollo)  is true
-- specification AG (decollo -> EF consegna)  is true
-- specification AG !(navigazione & hovering)  is true
-- specification AG (navigazione -> EF hovering)  is true
-- specification AG (hovering -> EF (posizionamento & EX atterraggio))  is true
-- specification AG (hovering -> E [ hovering U (consegna | posizionamento) ] )  is true
-- specification  G (decollo ->  X navigazione)  is true
-- specification  G (navigazione ->  X hovering)  is true
-- specification  G (consegna ->  X hovering)  is true
-- specification  G (posizionamento ->  X atterraggio)  is true
-- specification  G (hovering -> ( X consegna ->  X ( X hovering)))  is true
-- specification  G (atterraggio ->  F decollo)  is true
-- specification  G (hovering ->  X ((navigazione | consegna) | posizionamento))  is true

C:\Users\giaco\Desktop\Sicurezza Informatica\2_ANNO\Metodi Formali per la Sicurezza\NuSMV-2.6.0-win64\bin>
```

Risultati casi di non soddisfacimento

```
-- specification  G (decollo ->  X navigazione)  is false
-- as demonstrated by the following execution sequence
Trace Description: LTL Counterexample
Trace Type: Counterexample
  -- Loop starts here
  -> State: 1.1 <-
    state = s1
    decollo = TRUE
    navigazione = FALSE
    hovering = FALSE
    consegna = FALSE
    posizionamento = FALSE
    atterraggio = FALSE
  -> State: 1.2 <-
```

```
-- specification AG (hovering -> E [ hovering U (consegna | posizionamento) ] )  is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
  -> State: 3.1 <-
    state = s1
    decollo = TRUE
    navigazione = FALSE
    hovering = FALSE
    consegna = FALSE
    posizionamento = FALSE
    atterraggio = FALSE
  -> State: 3.2 <-
    state = s2
    decollo = FALSE
    navigazione = TRUE
  -> State: 3.3 <-
    state = s3
    navigazione = FALSE
    hovering = TRUE
```