



Università degli Studi di Bari 'Aldo Moro'
Dipartimento di Informatica
Corso di Laurea Magistrale in Sicurezza Informatica

CASO DI STUDIO
METODI FORMALI PER LA SICUREZZA
A.A. 2025/2026

Matteo Esposito

Contesto

L'evoluzione della logistica punta sui droni (UAV) per ridurre tempi e impatto ambientale.

Aziende come Amazon Prime Air stanno testando consegne in garantite 30 minuti con droni elettrici autonomi.

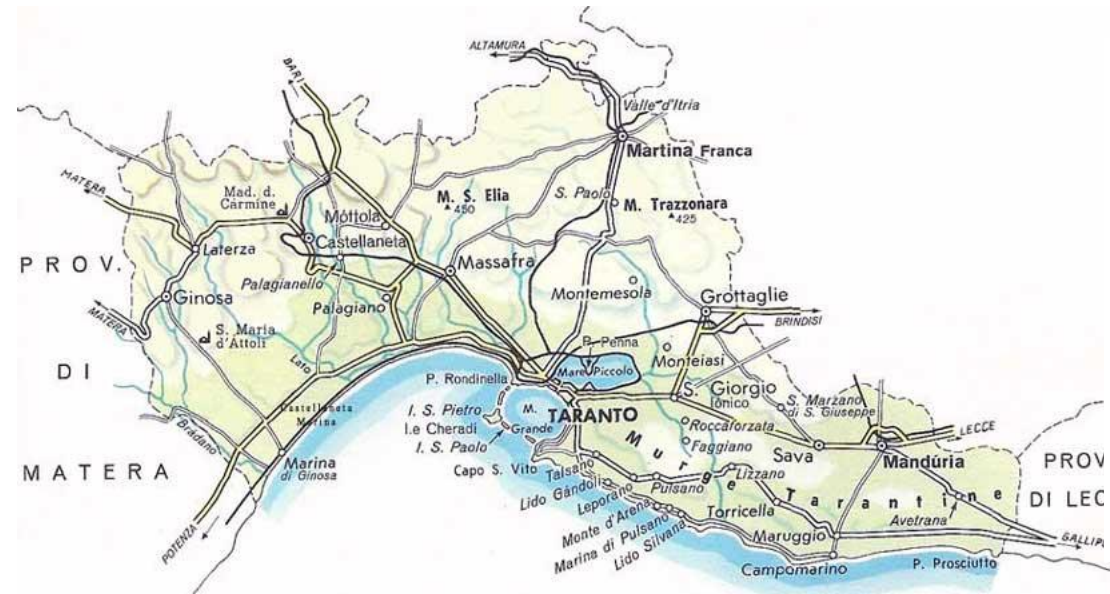


Caso di Studio

Modellazione di un sistema di Delivery
nella provincia di Taranto

Obiettivi:

- Ottimizzazione del percorso adattando il Problema del Commesso Viaggiatore per pianificare la sequenza di consegne
- Verifica formale del comportamento modellando il drone come una FSM per la correttezza delle manovre e verifica delle proprietà di sicurezza tramite specifiche LTL e CTL

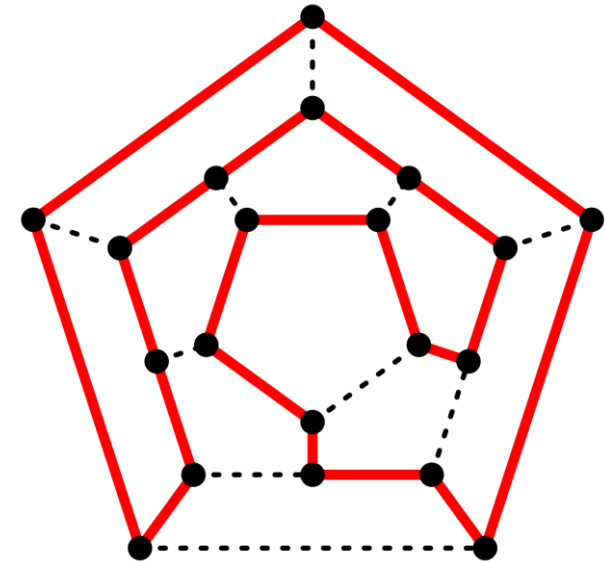


Caso di Studio

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.

Il problema in matematica si trasforma in una ricerca di un Ciclo Hamiltoniano di peso minimo in un grafo pesato

Il TSP è un problema NP-hard, di conseguenza non esiste un algoritmo efficiente che garantisca sempre la soluzione ottimale in tempi ragionevoli.



Caso di Studio

I sei nodi logistici identificati:

- Taranto: Punto di partenza e ricarica (15m s.l.m.)
- Pulsano (37m s.l.m.)
- Manduria(79m s.l.m.)
- Grottaglie (130m s.l.m.)
- Castellaneta (245m s.l.m.)
- Martina Franca (431m s.l.m.)



Funzione di costo

Il consumo della batteria è dato da:

- ***Costo Base = Distanza(in km)* 2***
- ***Costo Extra = Dislivello(in m)/ 10***

La formula implementata è:

- ***Costo Batteria = Costo Base + Costo Extra***

L'euristica stabilisce un rapporto di costo di circa 50:1.

Viene preso in considerazione solo il dislivello positivo.



Modellazione in ASP

Modelliamo i nodi:

```
% --- 3. DATI GEOGRAFICI ---  
% Alitudine dei nodi espressa in metri sul mare  
altitude(taranto, 15).  
altitude(martina, 431).  
altitude(grottaglie, 130).  
altitude(manduria, 79).  
altitude(castellaneta, 245).  
altitude(pulsano, 37).  
  
% Distanza tra i nodi espressa in chilometri  
dist_raw(taranto, martina, 30).  
dist_raw(taranto, grottaglie, 20).  
dist_raw(taranto, manduria, 35).  
dist_raw(taranto, castellaneta, 33).  
dist_raw(taranto, pulsano, 15).  
dist_raw(martina, grottaglie, 14).  
dist_raw(martina, manduria, 38).  
dist_raw(martina, castellaneta, 32).  
dist_raw(martina, pulsano, 40).  
dist_raw(grottaglie, manduria, 22).  
dist_raw(grottaglie, castellaneta, 45).  
dist_raw(grottaglie, pulsano, 25).  
dist_raw(manduria, pulsano, 20).  
dist_raw(manduria, castellaneta, 60).  
dist_raw(castellaneta, pulsano, 45).  
  
% Codice per evitare di riportare due volte la stessa distanza per il percorso inverso  
distance(X,Y,D) :- dist_raw(X,Y,D).  
distance(X,Y,D) :- dist_raw(Y,X,D).
```



Modellazione in ASP

Definiamo il ciclo Hamiltoniano:

```
% Obbligo di uscita
% Per ogni nodo, deve essere selezionato esattamente un arco uscente,
{ cycle(X,Y) : edge(X,Y) } = 1 :- node(X).
% Obbligo di entrata
% Per ogni nodo, deve essere selezionato esattamente un arco entrante
{ cycle(X,Y) : edge(X,Y) } = 1 :- node(Y).
```

Definiamo la funzione di costo:

```
% --- 4. FUNZIONE DI COSTO ---
% Viene considerato il dislivello tra due nodi solo se positivo, altrimenti é
% trascurabile
elevation_diff(X,Y, Diff) :- altitude(X, H1), altitude(Y, H2), H2 > H1, Diff = H2 -
H1.
elevation_diff(X,Y, 0) :- altitude(X, H1), altitude(Y, H2), H2 <= H1.

% Costo Batteria = (Distanza * 2) + (Dislivello / 10)
battery_cost(X,Y,B) :-
    distance(X,Y,D),
    elevation_diff(X,Y,Diff),
    BaseCost = D * 2,
    ClimbCost = Diff / 10,
    B = BaseCost + ClimbCost.

% Calcolo del costo totale (qui sommo distanza e batteria come indice di performance)
cost(X,Y,C) :- distance(X,Y,D), battery_cost(X,Y,B), C = D + B.
```



Modellazione in ASP

Verifichiamo che ogni nodo venga incluso nel ciclo:

```
% Verifica completezza del ciclo
cycle_complete :- N = #count { X : node(X) },
                  N = #count { X : reached(X) }.
:- not cycle_complete.
```

Script per la ottimizzazione del costo batteria:

```
% --- 6. OTTIMIZZAZIONE ---
% Minimizza il consumo di batteria
#minimize { B,X,Y : cycle(X,Y), battery_cost(X,Y,B) }.
```



Risultati

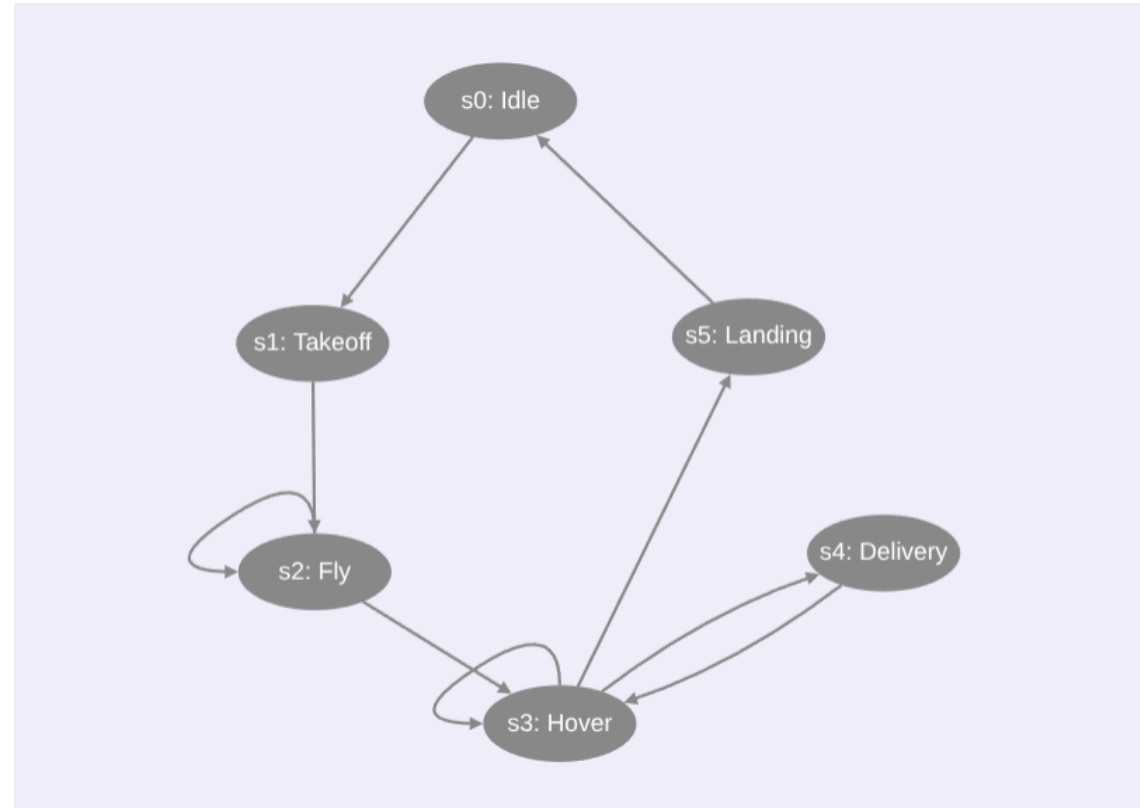
Percorso ottimale identificato:

Taranto, Pulsano, Manduria, Grottaglie, Martina Franca, Castellaneta, Taranto



Struttura di Kripke

- $S = \{s_0, s_1, s_2, s_3, s_4, s_5, s_6\}$
- $I = \{s_0\}$
- $R = \{(s_0, s_1), (s_1, s_2), (s_2, s_2), (s_2, s_3), (s_3, s_2), (s_3, s_3), (s_3, s_4), (s_4, s_3), (s_3, s_5), (s_5, s_0)\}$
- $L(s_0) = \{\text{'Idle'}\}$
- $L(s_1) = \{\text{'Takeoff'}\}$
- $L(s_2) = \{\text{'Fly'}\}$
- $L(s_3) = \{\text{'Hover'}\}$
- $L(s_4) = \{\text{'Delivery'}\}$
- $L(s_5) = \{\text{'Landing'}\}$



Codifica NuSMV

Variabili utilizzate:

```
• VAR
•   state : {idle, takeoff, fly, hover, delivery, landing};
•   payload : {yes, no};
```

Codifica degli stati e relazione tra di essi:

```
• ASSIGN
•   init(state) := idle;
•   init(payload) := yes;
•
•   next(state) := case
•     state = idle : takeoff;
•     state = takeoff : fly;
•     state = fly : {fly, hover};
•
•     state = hover & payload = yes : {delivery, fly, landing, hover};
•     state = hover & payload = no : {fly, landing, hover};
•
•     state = delivery : hover;
•     state = landing : idle;
•     TRUE : state;
•   esac;
•
•   next(payload) := case
•     state = delivery : {yes, no};
•     state = idle : yes;
•     TRUE : payload;
•   esac;
```

Vincoli di Fairness:

```
FAIRNESS !(state = fly)
FAIRNESS !(state = hover)
```



Proprietà LTL

LTLSPEC

```
G (state = fly -> (state = fly U state = hover));
```

- Il drone rimarrà in volo fino a che non entra in fase di hovering

LTLSPEC

```
G (state = delivery -> X state = hover);
```

- Dopo la consegna, il drone tornerà in fase di hovering

LTLSPEC

```
G (state = takeoff -> X state = fly);
```

- Successivamente al decollo il drone entrerà nello stato di volo

LTLSPEC

```
(payload = yes) U (state = delivery);
```

- Caso di non soddisfacimento, il carico deve restare a bordo finché il drone non lo consegna



Proprietà CTL

CTLSPEC

AG (state in {takeoff, fly, hover, delivery} -> EF (state = landing | state = idle));

- Garantisce che il drone non rimanga infinitamente in volo

CTLSPEC

AG (state = hover -> EF state = fly);

- Successivamente alla fase di hovering il drone potrebbe rimettersi in volo

CTLSPEC

AG (state = landing -> EF state = takeoff);

- Il drone potrebbe ripartire dopo un atterraggio.

CTLSPEC

AG ((state = hover & payload = yes) -> AF state = delivery);

- Caso di non soddisfacimento, Il drone obbligatoriamente deve consegnare se ha un pacco.



Risultati

```
-- specification AG (state in ((takeoff union fly) union hover) union delivery -> EF (state = landing | state = idle)) is true
-- specification AG (state = hover -> EF state = fly) is true
-- specification AG (state = landing -> EF state = takeoff) is true
-- specification AG ((state = hover & payload = yes) -> AF state = delivery) is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
-> State: 1.1 <-
  state = idle
  payload = yes
-> State: 1.2 <-
  state = takeoff
-> State: 1.3 <-
  state = fly
-- Loop starts here
-> State: 1.4 <-
  state = hover
-> State: 1.5 <-
  state = landing
-> State: 1.6 <-
  state = idle
-> State: 1.7 <-
  state = takeoff
-> State: 1.8 <-
  state = fly
-> State: 1.9 <-
  state = hover
-- specification G (state = fly -> (state = fly U state = hover)) is true
-- specification G (state = delivery -> X state = hover) is true
-- specification G (state = takeoff -> X state = fly) is true
-- specification (payload = yes U state = delivery) is false
-- as demonstrated by the following execution sequence
Trace Description: LTL Counterexample
Trace Type: Counterexample
-- Loop starts here
-> State: 2.1 <-
  state = idle
  payload = yes
-> State: 2.2 <-
  state = takeoff
-> State: 2.3 <-
  state = fly
-> State: 2.4 <-
  state = hover
-> State: 2.5 <-
  state = landing
-> State: 2.6 <-
  state = idle
```

