Università degli Studi di Bari Aldo Moro

Laurea Magistrale in Sicurezza Informatica

Metodi Formali per la Sicurezza

Sistema di consegne via Drone per la provincia di Taranto

Prof.ssa Francesca Alessandra Lisi

Dott. Matteo Esposito

1. Introduzione

L'evoluzione della logistica sta vedendo una rapida integrazione di sistemi autonomi, in particolare degli Unmanned Aerial Vehicle, comunemente noti come droni. L'utilizzo di droni per la consegna delle merci promette di ridurre drasticamente i tempi di spedizione e l'impatto ambientale legato al trasporto su gomma.

Un importante esempio di questa trasformazione è l'iniziativa Amazon Prime Air. Il colosso dell'e-commerce ha sviluppato un servizio volto a consegnare pacchi fino a 2,5 kg in meno di 30 minuti utilizzando droni completamente elettrici e autonomi. Progetti di tale portata evidenziano come la consegna via drone stia passando dalla fase sperimentale a quella operativa su larga scala.

L'implementazione di tali sistemi in scenari reali solleva due classi di problemi fondamentali che richiedono approcci rigorosi per essere risolti:

- Efficienza Operativa: La capacità di pianificare percorsi ottimali per minimizzare il consumo di batteria;
- Sicurezza: La garanzia che il drone operi secondo protocolli di manovra sicuri, evitando stati inconsistenti o pericolosi. Un errore software non comporta solo la perdita del drone, ma rischi per persone e cose a terra.

L'utilizzo di Metodi Formali per la certificazione di questi sistemi diventa necessario per garantire gli standard di affidabilità richiesti dal settore.

L'obiettivo del caso di studio è l'applicazione di Metodi Formali per modellare e verificare le logiche decisionali che governano tali sistemi.

Nello specifico, il progetto affronta due problemi distinti:

a. Ottimizzazione del percorso adattando il Problema del Commesso Viaggiatore per pianificare la sequenza di consegne. Si è scelto di ottimizzare il percorso in base al consumo stimato di batteria. Questa metrica è critica nei droni reali, dove l'autonomia è il vincolo principale.
b. Verifica formale del comportamento modellando il drone come una Macchina a Stati Finiti per verificare la correttezza delle manovre (decollo, hovering, atterraggio, consegna). Attraverso logiche temporali (LTL/CTL), verranno verificate proprietà di sicurezza.

Per il raggiungimento degli obiettivi, sono stati utilizzati:

- Answer Set Programming (ASP): Utilizzato per risolvere il TSP adattato alle necessità del caso di studio tramite il solver Clingo [1].

- Model Checking Simbolico: Utilizzato per la verifica delle manovre del drone tramite NuSMV [2] [3].

2. Ottimizzazione del Percorso

In questa sezione viene affrontata la risoluzione del problema di routing del drone. L'obiettivo è determinare la sequenza ottimale di consegne che minimizzi il consumo energetico, garantendo che ogni nodo del percorso venga servito una sola volta.

2.1 Il Problema del Commesso Viaggiatore (TSP)

Il Problema del Commesso Viaggiatore [4] è uno dei problemi più noti nell'ambito dell'ottimizzazione combinatoria. La sua definizione è apparentemente semplice: dato un insieme di città e le distanze tra ogni coppia di esse, l'obiettivo è trovare il percorso più breve che visiti ogni città esattamente una volta e ritorni al punto di partenza.

Matematicamente, il TSP può essere modellato utilizzando la teoria dei grafi. Il problema richiede di trovare un Ciclo Hamiltoniano di peso minimo in un grafo pesato.

Un Ciclo Hamiltoniano è un percorso chiuso che tocca ogni vertice del grafo esattamente una volta.

La difficoltà del TSP risiede nella sua natura NP-hard. Il numero di possibili percorsi cresce fattorialmente con il numero di nodi (N!). Se per pochi nodi la soluzione è calcolabile manualmente, al crescere di N diventa computazionalmente intrattabile per un calcolo a forza bruta. Per questo motivo, l'utilizzo di strumenti come l'Answer Set Programming (ASP) risulta efficace per definire i vincoli e lasciare al solver il compito di esplorare lo spazio di ricerca in modo efficiente.

2.2 Scenario del Caso di Studio

In questo progetto, il classico TSP è stato adattato a uno scenario di logistica reale: la consegna di pacchi tramite drone nella provincia di Taranto. A differenza di un classico TSP che minimizza solo la distanza, questo modello mira a minimizzare il consumo energetico totale della batteria, considerando anche l'altitudine dei nodi.

L'area di interesse comprende nodi con caratteristiche orografiche molto diverse, passando dal livello del mare a zone collinari. Questa eterogeneità rende il problema non banale, poiché il percorso più breve in linea d'aria non è necessariamente il più efficiente dal punto di vista energetico.

La rete logistica è composta da sei nodi:

- Taranto (Hub): Punto di partenza e ricarica (15m s.l.m.);
- Pulsano: Località costiera pianeggiante (37m s.l.m.);
- Manduria: Entroterra pianeggiante (79m s.l.m.);

- Grottaglie: Zona di media altitudine (130m s.l.m.);
- Castellaneta: Zona delle gravine (245m s.l.m.);
- Martina Franca: Punto critico in Valle d'Itria ad alta quota (431m s.l.m.).

2.3 Funzione di costo

Il consumo della batteria è stato modellato considerando due componenti:

1. Costo Base: L'energia necessaria per coprire la distanza orizzontale.

Costo Base = Distanza * 2

2. Costo Extra: L'energia supplementare necessaria per vincere la forza di gravità e guadagnare quota. Viene considerato solo il dislivello positivo.

Costo Extra = Dislivello / 10

La formula implementata è:

Costo Batteria = Costo Base + Costo Extra

Combinando unità diverse, l'euristica stabilisce un rapporto di costo di circa 50:1: salire di 1 metro costa all'algoritmo quanto percorrere 50 metri in piano.

2.4 Modellazione in ASP

Il codice è stato scritto in linguaggio ASP ed eseguito tramite il solver Clingo. Di seguito vengono analizzate le sezioni chiave della codifica.

Il grafo è definito come completo (ogni nodo è collegato a tutti gli altri). La regola di generazione dei cicli utilizza i vincoli di cardinalità per imporre la struttura del 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).
```

I dati sono stati inseriti rispecchiando la realtà geografica della provincia.

```
% --- 3. DATI GEOGRAFICI ---
% Alititudine 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).
```

Di seguito viene riportato lo script per il calcolo della 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.
```

Il seguente script verifica che tutti i nodi vengano inclusi nel ciclo:

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

Infine, l'istruzione di ottimizzazione che minimizza il costo batteria.

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

2.5 Risultati

Eseguendo il codice sul solver Clingo, il sistema ha esplorato lo spazio delle soluzioni calcolando diversi percorsi validi e convergendo verso l'ottimo globale.

```
clingo version 5.7.2 (6bd7584d)
Reading from stdin
Solving...
Answer: 1
cycle(taranto,manduria) cycle(martina,grottaglie) cycle(grottaglie,pulsano) cycle(manduria,castellaneta)
cycle(pulsano,taranto) cycle(castellaneta,martina) total_battery(402) total_distance(181)
Optimization: 402
Answer: 2
cycle(taranto,pulsano) cycle(martina,grottaglie) cycle(grottaglie,castellaneta) cycle(castellaneta,taranto)
cycle(manduria,martina) cycle(pulsano,manduria) total_battery(382) total_distance(165)
Optimization: 382
Answer: 3
cycle(taranto,pulsano) cycle(martina,grottaglie) cycle(grottaglie,manduria) cycle(manduria,taranto)
cycle(castellaneta,martina) cycle(pulsano,castellaneta) total_battery(366) total_distance(163)
Optimization: 366
Answer: 4
cycle(taranto,pulsano) cycle(martina,grottaglie) cycle(manduria,castellaneta) cycle(grottaglie,taranto)
cycle(castellaneta,martina) cycle(pulsano,manduria) total_battery(362) total_distance(161)
Optimization: 362
Answer: 5
cycle(taranto,pulsano) cycle(martina,castellaneta) cycle(castellaneta,taranto) cycle(grottaglie,martina)
cycle(manduria,grottaglie) cycle(pulsano,manduria) total_battery(313) total_distance(136)
Optimization: 313
OPTIMUM FOUND


Models       : 5
  Optimum    : yes
Optimization : 313
Calls        : 1
Time         : 0.383s (Solving: 0.09s 1st Model: 0.00s Unsat: 0.06s)
CPU Time     : 0.000s
```

Ricostruendo la catena dei predicati, il percorso suggerito è il seguente:

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

Emerge chiaramente come l'algoritmo abbia migliorato l'efficienza energetica andando a servire prima le località più vicine alla costa e di conseguenza con un'altitudine minore (Pulsano e Manduria) per poi servire i nodi con un’altitudine maggiore (Grottaglie, Martina Franca e Castellaneta).

3. Comportamento del Drone

Definiamo innanzitutto gli stati del drone e la struttura logica sottostante. Gli stati identificati per il nostro modello sono i seguenti:

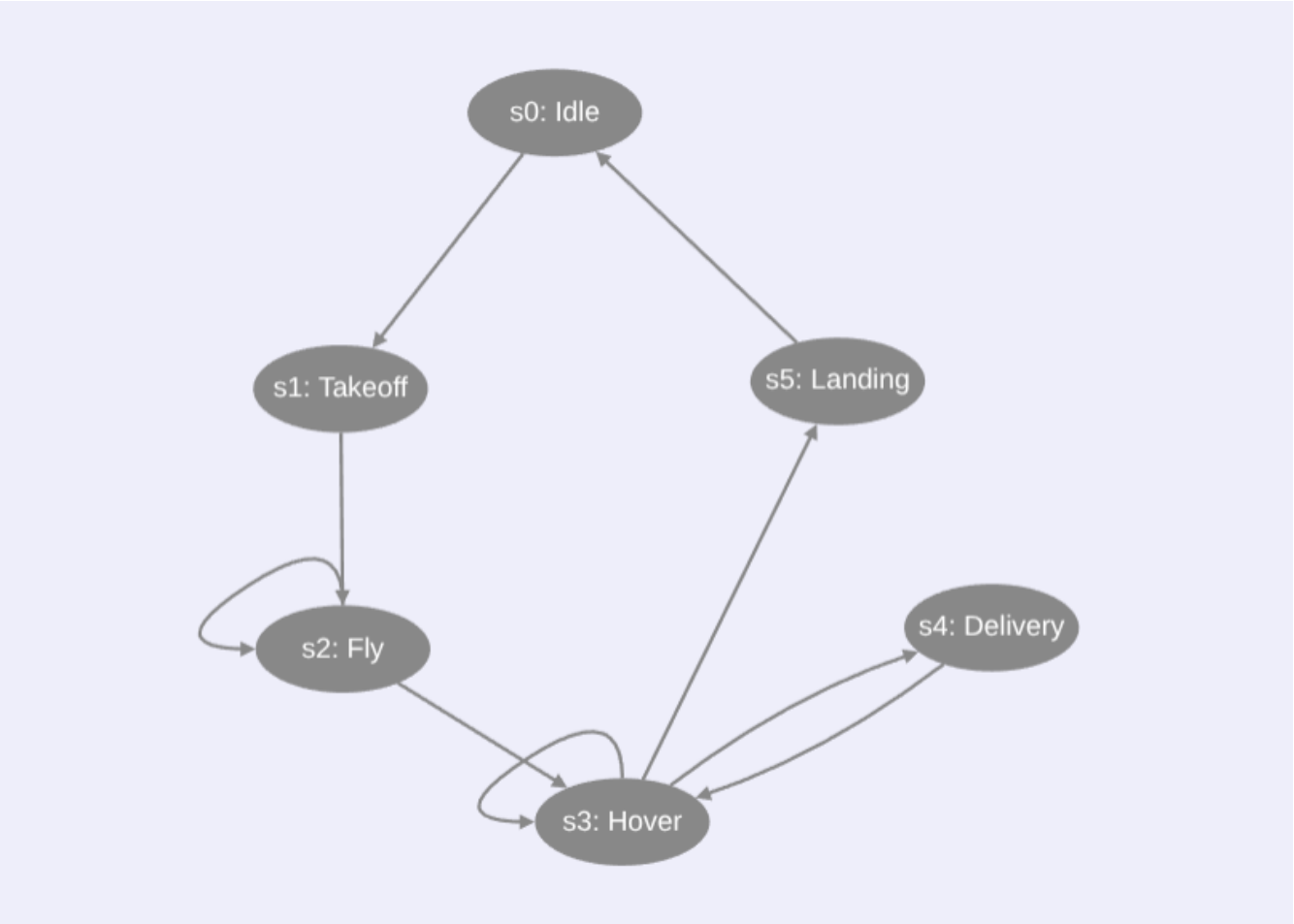
- Idle: Stato di riposo a terra, con motori spenti o al minimo;
- Takeoff: Fase di decollo, in cui il drone acquisisce quota per iniziare la missione;
- Fly: Il drone si sposta verso il punto di interesse. In questo stato il drone può impiegare più unità di tempo;
- Hover: Stato di decisione. Il drone è stabile sulla destinazione e deve decidere se procedere alla consegna, attendere, o abortire la missione atterrando;
- Delivery: Fase di rilascio del carico;
- Landing: Fase di discesa verso terra per concludere la missione o in caso di aborto.

Nel modello è stata introdotta la variabile di stato payload, che discrimina se il drone ha ancora il pacco a bordo o se è stato consegnato.

3.1 Struttura di Kripke

La semantica del sistema è interpretata mediante una Struttura di Kripke, definita come segue:

- Insieme degli stati S = {s0, s1, s2, s3, s4, s5, s6}
- Insieme degli stati Iniziali I = {s0}
- Insieme relazione sugli stati di transizione R = {(s0, s1), (s1, s2), (s2, s2), (s2, s3), (s3, s2), (s3, s3), (s3, s4), (s4, s3), (s3, s5), (s5, s0)}
- Funzione di etichettatura L
    - L(s0) = {‘Idle’}
    - L(s1) = {‘Takeoff’}
    - L(s2) = {‘Fly’}
    - L(s3) = {‘Hover’}
    - L(s4) = {‘Delivery’}
    - L(s5) = {‘Landing’}



3.2 Codifica NuSMV

Il sistema è definito da due variabili di stato principali:

- state: Variabile enumerativa che rappresenta lo stato corrente del drone. Il dominio comprende: {idle, takeoff, fly, hover, delivery, landing}.
- payload: Variabile enumerativa {yes, no} che indica la presenza del carico a bordo.

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

Il costrutto next(state) := case definisce le regole di evoluzione del sistema ad ogni passo temporale. Le transizioni implementate sono le seguenti:

- Fase di Volo**:** Lo stato fly prevede una transizione ricorsiva, dove il drone può rimanere in volo per un tempo indefinito prima di raggiungere la destinazione.

- Fase decisionale: Lo stato di hover rappresenta il punto nevralgico del modello. Da qui, il drone può effettuare diverse scelte operative:
    - Effettuare la consegna (Delivery);
    - Ritornare in volo;
    - Atterrare per completamento o per abortire la missione.

```
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;
```

Un aspetto critico della codifica è l'introduzione dei vincoli di fairness. Senza questi vincoli, il Model Checker potrebbe generare falsi controesempi basati su tracce irrealistiche in cui il drone rimane infinitamente nello stato di volo o di hovering. La direttiva FAIRNESS istruisce NuSMV a scartare tali percorsi, considerando valide solo le esecuzioni in cui il sistema, prima o poi, esce da questi stati.

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

3.3 Proprietà LTL

Le proprietà LTL verificano il comportamento del sistema su percorsi lineari infiniti. Le specifiche implementate sono:

```
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.

3.4 Proprietà CTL

Le proprietà CTL esplorano le ramificazioni delle possibili esecuzioni future. Vengono riportate di seguito le specifiche implementate:

```
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.

3.5 Risultati

Di seguito si riportano gli output dell'esecuzione di NuSMV che confermano l'analisi.

```
-- 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
```

L'analisi condotta tramite NuSMV ha confermato la piena validità delle tre proprietà LTL e delle tre proprietà CTL. Per i due casi di non soddisfacimento introdotti intenzionalmente:

- Per quanto concerne la logica lineare (LTL), la specifica imponeva la permanenza del carico a bordo fino alla consegna obbligatoria è risultata falsa; l'analisi della traccia di controesempio generata dal model checker mostra infatti una sequenza in cui il drone, nonostante abbia raggiunto la fase di hover, transita nello stato di atterraggio senza effettuare il rilascio. Questo comportamento è corretto, in quanto dimostra che il sistema potrebbe abortire la missione.

- Per la specifica CTL, è stata confutata l'ipotesi di inevitabilità della consegna: il controesempio evidenzia come, pur trovandosi in *hover* con il payload, il drone non sia vincolato a un unico futuro che porta al rilascio del pacco, ma potrebbe decidere di optare per un rientro alla base o per la ripresa della navigazione, confermando che il completamento della missione è solo una possibilità.

4. Bibliografia

1. Documentation [WWW Document], n.d. URL https://potassco.org/doc/ (accessed 1.19.26).
2. Cimatti, A., Clarke, E., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., Tacchella, A., 2002. NuSMV 2: An OpenSource Tool for Symbolic Model Checking, in: Lecture Notes in Computer Science. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 359–364.
3. NuSMV home page [WWW Document], n.d. URL http://nusmv.fbk.eu/ (accessed 1.19.26).
4. https://www.sciencedirect.com/topics/computer-science/traveling-salesman-problem