SMART-TRIP: Sistema Multi-paradigma per l'Analisi e Raccomandazione di Tragitti Intelligenti per Pianificazione

Gruppo di lavoro

Antonio Colamartino, 778730, a.colamartino6@studenti.uniba.it

Repository: https://github.com/Tony0380/Smart_Trip

Anno Accademico: 2024-2025

Indice

Capitolo 0) Introduzione

Il dominio di interesse del progetto **SMART-TRIP** riguarda l'ottimizzazione intelligente della pianificazione di viaggi attraverso l'integrazione di paradigmi eterogenei dell'Intelligenza Artificiale. Il sistema affronta il problema complesso della ricerca di percorsi ottimali su reti di trasporto urbano, considerando molteplici criteri simultanei (costo, tempo, comfort) e personalizzazione basata su profili utente.

Il dominio presenta sfide significative che richiedono l'applicazione coordinata di diverse tecniche di AI:

- **Complessità computazionale** nella ricerca di percorsi ottimi su grafi pesati multi-dimensionali
- **Incertezza** nelle previsioni di costi e tempi di viaggio dovuta a fattori esterni (meteo, traffico, stagionalità)
- **Personalizzazione** delle raccomandazioni basata su preferenze individuali e profili comportamentali
- **Vincoli logici** complessi derivanti da regolamentazioni, disponibilità e constraints di business

Sommario del Sistema

Il sistema **SMART-TRIP** implementa un Knowledge-Based System (KBS) ibrido che integra quattro paradigmi complementari dell'Intelligenza Artificiale per fornire soluzioni ottimali personalizzate nel dominio della pianificazione viaggi.

Il sistema orchestra moduli specializzati che dimostrano competenze specifiche su diversi argomenti del programma:

- Un **modulo di ricerca** implementa algoritmi di ottimizzazione su grafi per il pathfinding
- Un **modulo di machine learning** gestisce predizione e classificazione con validazione rigorosa
- Un **modulo probabilistico** modella l'incertezza attraverso reti bayesiane
- Un **modulo logico** codifica vincoli e regole di business in Prolog

L'integrazione multi-paradigma avviene attraverso un orchestratore centrale che coordina l'elaborazione sequenziale dei moduli, aggregando i risultati in raccomandazioni finali con spiegazioni interpretabili.

Elenco argomenti di interesse

Argomento 1: Algoritmi di Ricerca (Sezione "Ricerca e Ottimizzazione" del programma) Implementazione di algoritmi di ricerca informata (A*) e non informata (Floyd-Warshall, Dijkstra) per ottimizzazione multi-criterio su grafi pesati rappresentanti reti di trasporto urbano.

Argomento 2: Apprendimento Automatico (Sezione "Apprendimento Automatico" del programma) Modelli di apprendimento supervisionato con Grid Search per predizione di variabili continue (prezzi, tempi) e classificazione di profili utente, con metodologie di validazione K-fold cross-validation.

Argomento 3: Ragionamento Probabilistico (Sezione "Ragionamento Probabilistico e Incertezza" del programma) Rete Bayesiana per modellazione dell'incertezza nelle decisioni di viaggio, gestendo dipendenze probabilistiche tra variabili ambientali e outcome di viaggio.

Argomento 4: Rappresentazione della Conoscenza (Sezione "Rappresentazione della Conoscenza e Ragionamento Automatico" del programma) Knowledge Base Prolog per codifica di vincoli logici, regole di

business e constraint satisfaction nella validazione di piani di viaggio.

Capitolo 1) Algoritmi di Ricerca

Sommario

Il modulo di ricerca implementa una rappresentazione basata su **grafi pesati multi-dimensionali** utilizzando NetworkX per modellare reti di trasporto urbano realistiche. La knowledge base spaziale è costituita da un grafo diretto dove i nodi rappresentano 20 città italiane principali con coordinate GPS reali e gli archi codificano connessioni di trasporto con attributi multipli (distanza geografica, costo economico, tempo di percorrenza, livello di comfort).

Il sistema utilizza coordinate geografiche reali estratte da OpenStreetMap per garantire accuratezza nelle distanze calcolate. Le città incluse spaziano geograficamente da Milano (45.4642°N, 9.1900°E) nel nord fino a Palermo (38.1157°N, 13.3613°E) nel sud, coprendo un'area di circa 1200 km di estensione geografica.

La rappresentazione della conoscenza geografica utilizza:

- **Coordinate geografiche** (latitudine, longitudine) per calcoli di distanza haversine reale
- **Matrice di adiacenza pesata** pre-computata con Floyd-Warshall per lookup O(1)
- **Funzioni euristiche ammissibili** basate su distanza haversine per A*
- **Attributi di trasporto realistici** derivati da dati empirici di velocità e costi medi

L'architettura del modulo separa chiaramente la fase di **pre-processing** (costruzione grafo e calcolo Floyd-Warshall) dalla fase di **query** (esecuzione algoritmi A* e Dijkstra), ottimizzando le performance per applicazioni interattive. Il pre-processing calcola tutte le 400 distanze possibili (20×20) una volta sola all'inizializzazione, mentre le query successive operano su strutture dati ottimizzate.

Strumenti utilizzati

Algoritmi di ricerca implementati:

- *A Multi-Objective** [1]: Versione estesa dell'algoritmo A* classico per ottimizzazione simultanea di multipli criteri (costo, tempo, distanza, comfort) tramite weighted sum approach. La funzione euristica utilizza la distanza haversine come lower bound ammissibile per tutti gli obiettivi.
- **Floyd-Warshall** [1]: Algoritmo di programmazione dinamica implementato per pre-calcolo di tutte le distanze minime tra ogni coppia di città. La complessità $O(n^3)$ è accettabile per grafi di dimensione 20 nodi, garantendo lookup O(1) per query future.
- **Dijkstra classico**: Utilizzato come baseline per confronti di performance, implementato con NetworkX per consistenza con il framework principale.
- **Beam Search**: Algoritmo di ricerca euristica con pruning che mantiene solo i migliori K candidati ad ogni livello (beam_width=3), offrendo un trade-off tra completezza ed efficienza computazionale.

Librerie e framework utilizzati:

- **NetworkX 3.0+** [4]: Libreria Python per manipolazione e analisi di grafi complessi, scelta per la robustezza degli algoritmi implementati e l'integrazione con l'ecosistema scientifico Python
- **NumPy**: Per calcoli matematici vettorizzati nelle funzioni di distanza haversine
- **Heapq**: Struttura dati heap per implementazione efficiente della priority queue in A*

Considerazioni implementative:

L'implementazione degli algoritmi segue fedelmente le specifiche teoriche classiche, con ottimizzazioni specifiche per il dominio viaggi:

- **Caching intelligente** delle distanze calcolate per ridurre computational overhead
- **Early termination** in A* quando il goal è raggiunto prima di esplorare tutto lo spazio
- **Gestione robusta** dei casi limite (nodi isolati, percorsi non esistenti)

Il modulo `pathfinder.py` implementa la classe `AdvancedPathfinder` che coordina tutti gli algoritmi, offrendo un'interfaccia unificata per il resto del sistema. La separazione tra algoritmi permette facilità di testing e benchmark comparativi.

Decisioni di Progetto

*Architettura Multi-Objective A:**

La scelta di estendere A* per ottimizzazione multi-criterio deriva dalla necessità di bilanciare obiettivi conflittuali nel dominio viaggi. L'implementazione utilizza il **weighted sum approach** che combina linearmente i diversi obiettivi:

- **Funzione di costo combinata**: `f(n) = g(n) + h(n)` dove `g(n) = w1·cost + w2·time + w3·distance + w4·comfort`
- **Pesi dinamici**: Derivati in tempo reale dalla classificazione ML del profilo utente tramite il modulo `UserProfileClassifier`
- **Euristica ammissibile**: La distanza haversine fornisce un lower bound valido per tutti gli obiettivi, garantendo l'ottimalità dell'algoritmo

La **normalizzazione** dei pesi avviene secondo la formula: `normalized_score = (1/(1+distance/1000)) * 0.3 + (1/(1+time/10)) * 0.3 + (1/(1+cost/100)) * 0.2 + comfort * 0.2`

Configurazione Floyd-Warshall:

La decisione di pre-computare Floyd-Warshall all'inizializzazione è motivata dall'analisi del trade-off spazio-tempo:

- **Pre-processing completo**: Matrice 20×20 (400 celle) richiede ~3.2KB di memoria ma garantisce lookup O(1)
- **Convergenza**: Utilizza epsilon = 1e-10 per gestire errori di floating-point nelle iterazioni k-i-j
- **Memory optimization**: La lookup table viene mantenuta in memoria per l'intera sessione, riducendo latenza delle query ripetute

Il **pattern di accesso** ottimizzato utilizza: `distances[origin][destination]` invece di ricerca su grafo, riducendo da O(|V| + |E|) a O(1) per ogni query di distanza.

Parametri di ottimizzazione per profili utente:

```
# Estratti dal codice reale (transport_profiles in pathfinder.py)
TRANSPORT_PROFILES = {
    'train': {'speed_kmh': 120, 'cost_per_km': 0.15, 'comfort': 0.8},
    'bus': {'speed_kmh': 80, 'cost_per_km': 0.08, 'comfort': 0.4},
```

```
    'flight': {'speed_kmh': 500, 'cost_per_km': 0.25, 'comfort': 0.6}
}

# Pesi per profili derivati da analisi comportamentale
PROFILE_WEIGHTS = {
    'business': {'cost': 0.2, 'time': 0.5, 'distance': 0.15, 'comfort': 0.15},
    'leisure': {'cost': 0.35, 'time': 0.25, 'distance': 0.25, 'comfort': 0.15},
    'budget': {'cost': 0.7, 'time': 0.2, 'distance': 0.05, 'comfort': 0.05}
}
```

Gestione connettivita grafo:

Il grafo implementa una **topologia realistica** che rispecchia la rete di trasporti italiana:

- **Collegamenti Nord**: Tutte le città del nord sono interconnesse via treno/bus
- **Collegamenti lunghe distanze**: Solo voli per tratte >600km (es. Milano-Palermo)
- **Hub strategici**: Roma e Milano fungono da nodi centrali con alta connettività

La **validazione dell'euristica** garantisce ammissibilità: per ogni nodo n e goal g, h(n) ≤ costo_reale(n,g), mantenendo l'ottimalità di A*.

Valutazione

Le metriche di valutazione seguono standard per algoritmi di pathfinding:

Metriche di Qualità:

- **Optimality Gap**: Deviazione dalla soluzione ottima Floyd-Warshall
- **Path Length**: Numero di nodi nel percorso trovato
- **Total Cost**: Somma pesata dei criteri ottimizzati

Metriche di Performance:

- **Computation Time**: Tempo di esecuzione medio
- **Nodes Expanded**: Nodi esplorati durante la ricerca
- **Memory Usage**: Consumo di memoria durante l'esecuzione

Algoritmo	Tempo Medio (ms)	Nodi Esplorati	Optimality Gap (%)
Floyd-Warshall	0.05	20^3	0.0
A* Multi-Obj	1.2	8.4	2.3
Dijkstra	0.8	12.1	0.0

I risultati mostrano che Floyd-Warshall garantisce ottimalità con overhead computazionale accettabile per grafi di dimensione limitata, mentre A* offre un buon compromesso tra velocità e qualità con euristica informata.

Capitolo 2) Apprendimento Automatico

Sommario

Il modulo di machine learning implementa una pipeline completa di **feature engineering** e **modellazione predittiva** per tre task principali: predizione prezzi, stima tempi di viaggio e classificazione profili utente. La knowledge base dei dati è costituita da dataset sintetici realistici con 29+ features che codificano attributi spaziali, temporali, economici e comportamentali derivati dal dominio viaggi.

Il sistema di machine learning si compone di tre moduli specializzati:

1. TravelPricePredictor: Modello di regressione per predizione dinamica dei prezzi di trasporto, che integra features geografiche (coordinate GPS, distanze), temporali (stagione, giorno della settimana, fasce orarie), economiche (domanda di base, fattori moltiplicativi) e comportamentali (profilo utente, sensibilità al prezzo).

2. UserProfileClassifier: Modello di classificazione supervisionata per identificazione automatica del profilo utente (business/leisure/budget) basato su caratteristiche demografiche e preferenze comportamentali.

3. TravelTimeEstimator: Modello di regressione per stima dei tempi di viaggio realistici che integra fattori ambientali spesso ignorati dagli algoritmi base (condizioni meteo, traffico, eventi speciali).

La rappresentazione della conoscenza predittiva utilizza:

- **Feature space multi-dimensionale** con 29 features per regressione prezzi, 15 features per classificazione profili
- **Encoding robusto** delle variabili categoriche tramite LabelEncoder con gestione di categorie non viste
- **Cross-validation stratificata K-fold** (K=5) per validazione rigorosa con bilanciamento classi
- **Grid Search sistematico** per ottimizzazione automatica degli iperparametri con 3-fold internal CV
- **Feature scaling** differenziato per algoritmi lineari (StandardScaler) vs tree-based (raw features)

Architettura del Feature Engineering:

Il sistema implementa una pipeline robusta di trasformazione dati che converte informazioni del dominio viaggi in rappresentazioni numeriche ottimizzate per machine learning:

- **Features geografiche**: Coordinate GPS, distanze haversine calcolate, regioni geografiche
- **Features temporali**: Encoding ciclico per stagioni, one-hot per giorni settimana, binario per weekend/peak hours
- **Features economiche**: Domanda normalizzata, moltiplicatori stagionali, indici di costo regionale
- **Features utente**: Profilo demografico (età, reddito), preferenze comportamentali (sensibilità prezzo/tempo/comfort)
- **Features contestuali**: Condizioni meteo, eventi speciali, fattori di traffico

Strumenti utilizzati

Modelli di Regressione per Price Prediction:

- **Linear Regression**: Modello baseline che assume relazioni lineari tra features e target, utilizzato per stabilire performance minima accettabile. Richiede StandardScaler per normalizzazione features.
- **Ridge Regression (L2)**: Estensione lineare con regolarizzazione L2 per controllo overfitting tramite penalty term $\lambda||w||^2$. Grid search su $\alpha \in [0.1, 1.0, 10.0, 100.0]$.
- **Random Forest Regressor**: Ensemble di alberi di decisione con bootstrap aggregating per gestione non-linearità. Parametri ottimizzati: n_estimators ∈ [50,100,150,200], max_depth ∈ [None,5,10,15],

min_samples_split ∈ [2,5,10].
- **Gradient Boosting Regressor**: Ensemble sequenziale che addestra weak learners su residui del predittore precedente. Parametri: learning_rate ∈ [0.01,0.1,0.2], n_estimators ∈ [50,100,150], max_depth ∈ [3,5,7].

Modelli di Classificazione per User Profiling:

- **Logistic Regression**: Classificatore lineare probabilistico con funzione softmax per output multi-classe. Fornisce probabilità di appartenenza calibrate per ogni profilo (business/leisure/budget).
- **Random Forest Classifier**: Ensemble di alberi per classificazione con voting majority. Robusto a outlier e capable di catturare interazioni complesse tra features demografiche.
- **Support Vector Classifier (SVM)**: Algoritmo di massima margine con kernel RBF per separazione non-lineare nello spazio delle features. Parametri C e gamma ottimizzati via grid search.

Modello Specializzato per Time Estimation:

- **Random Forest Regressor dedicato**: Modello separato addestrato specificamente per predizione tempi con features ridotte (distanza, tipo trasporto, condizioni meteo, ora di punta, eventi speciali).

Framework e Pipeline:

- **Scikit-learn 1.3+** [2]: Libreria principale per algoritmi ML, grid search, cross-validation e metriche di valutazione
- **Pandas**: Manipolazione dataset e feature engineering con DataFrame ad alta performance
- **NumPy**: Calcoli numerici vettorizzati per trasformazioni features
- **Joblib**: Serializzazione modelli trainati per persistenza e riutilizzo

Metodologie di Validazione:

- **K-Fold Cross-Validation** (K=5): Divisione dataset in 5 fold con training su 4 fold e validazione su 1, ripetuto 5 volte per robustezza statistica
- **Stratified Split**: Mantenimento distribuzione target classes nei fold per classificazione bilanciata
- **GridSearchCV**: Ricerca esaustiva su griglia parametri con cross-validation interna per evitare overfitting
- **Train/Validation/Test Split**: 60%/20%/20% per separazione rigorosa con test set mai visto durante development

Decisioni di Progetto

Architettura Feature Engineering:

La progettazione del feature engineering è guidata da domain expertise nel settore viaggi, con l'obiettivo di catturare tutti i fattori che influenzano prezzi e comportamenti utente:

```
# Feature set per predizione prezzi (estratto dal codice reale)
FEATURE_GROUPS = {
    'geographic': ['distance', 'origin_lat', 'origin_lon', 'dest_lat',
'dest_lon'],
    'temporal': ['day_of_week', 'hour', 'is_weekend', 'is_peak_hour'],
    'user': ['user_age', 'user_income', 'price_sensitivity', 'time_priority',
'comfort_priority'],
```

```
    'transport': ['available_train', 'available_bus', 'available_flight'],
    'contextual': ['base_demand', 'seasonal_multiplier', 'weather_factor',
'special_event']
}
```

Configurazione Grid Search Ottimizzata:

Il grid search utilizza una strategia gerarchica per ridurre il computational cost mantenendo copertura completa dello spazio parametri:

```
PARAM_GRIDS = {
    'ridge': {'alpha': [0.1, 1.0, 10.0, 100.0]},  # Range logaritmico
    'random_forest': {
        'n_estimators': [50, 100, 150, 200],
        'max_depth': [None, 5, 10, 15],
        'min_samples_split': [2, 5, 10]
    },
    'gradient_boosting': {
        'n_estimators': [50, 100, 150],
        'learning_rate': [0.01, 0.1, 0.2],
        'max_depth': [3, 5, 7]
    }
}
```

Decisioni di Feature Engineering:

- **Encoding Categorico Robusto**: Utilizzo di LabelEncoder con fallback handling per categorie non viste durante inference, garantendo robustezza in produzione
- **Normalizzazione Selettiva**: StandardScaler applicato solo a modelli lineari (Linear/Ridge Regression), mentre tree-based models utilizzano raw features per preservare interpretabilità
- **Feature Selection Conservativa**: Mantenimento di tutte le 29 features originali dopo analisi di correlazione (nessuna feature con |r| > 0.95)
- **Temporal Feature Engineering**: Encoding delle features temporali per catturare periodicità (giorno settimana, stagioni) senza perdita informazione

Architettura di Integrazione ML-Search:

Una decisione progettuale chiave è l'**integrazione dinamica** tra modelli ML e algoritmi di ricerca. I modelli ML non operano in isolamento ma forniscono input real-time agli algoritmi A*:

```
# Pipeline integrazione (da ml_pathfinder_integration.py)
def find_ml_enhanced_route(self, origin, destination, **context):
    # 1. User profiling ML
    profile, weights = self.classify_user_profile(**context)

    # 2. Price prediction ML per ogni edge
    enhanced_graph = self.enhance_graph_with_ml_predictions(...)
```

```
        # 3. A* search con pesi personalizzati
        route = self.pathfinder.multi_objective_astar(weights=weights)

        return route, metadata
```

Strategia di Validazione Multi-livello:

- **Inner CV**: 3-fold cross-validation all'interno del GridSearchCV per selezione iperparametri
- **Outer CV**: 5-fold cross-validation per valutazione finale dei modelli selezionati
- **Temporal Holdout**: Test set chronologically separato per simulare deployment scenario
- **Cross-Model Validation**: Confronto sistematico tra tutti gli algoritmi con paired t-test per significatività statistica

Gestione dell'Overfitting:

- **Regularization**: Utilizzo sistematico di L2 penalty (Ridge) e tree pruning (Random Forest)
- **Early Stopping**: Monitoraggio validation loss per Gradient Boosting con tolerance=1e-4
- **Feature Stability**: Analisi feature importance across fold per identificare features instabili
- **Learning Curves**: Validazione empirica che training/validation error convergono

Valutazione

Metriche per Regressione:

- **R^2 Score**: Coefficiente di determinazione per explained variance
- **Mean Absolute Error (MAE)**: Errore assoluto medio in unità originali
- **Root Mean Squared Error (RMSE)**: Penalizzazione errori quadratici

Risultati Predizione Prezzi:

Modello	R^2 Score	MAE (€)	RMSE (€)
Linear Regression	0.889 ± 0.012	8.5 ± 0.8	12.1 ± 1.2
Ridge Regression	0.859 ± 0.015	9.5 ± 0.9	13.4 ± 1.1
Random Forest	0.750 ± 0.023	11.3 ± 1.2	16.8 ± 1.8
Gradient Boosting	**0.908 ± 0.010**	**6.8 ± 0.6**	**10.3 ± 0.9**

Metriche per Classificazione:

- **Accuracy**: Frazione predizioni corrette
- **Precision/Recall**: Metriche per classi sbilanciate
- **F1-Score**: Media armonica precision-recall

Risultati Classificazione Profili:

Modello	Accuracy	Precision	Recall	F1-Score
Logistic Regression	0.95 ± 0.02	0.94 ± 0.03	0.95 ± 0.02	0.94 ± 0.02

Modello	Accuracy	Precision	Recall	F1-Score
Random Forest	**1.00 ± 0.00**	**1.00 ± 0.00**	**1.00 ± 0.00**	**1.00 ± 0.00**
SVM	0.92 ± 0.03	0.91 ± 0.04	0.92 ± 0.03	0.91 ± 0.03

I risultati mostrano performance eccellenti per entrambi i task, con Gradient Boosting ottimale per regressione (R^2=0.908) e Random Forest perfetto per classificazione (Accuracy=1.00).

Capitolo 3) Ragionamento Probabilistico

Sommario

Il modulo probabilistico implementa una **Rete Bayesiana** per modellazione esplicita dell'incertezza e supporto alle decisioni nel dominio viaggi. La rappresentazione della conoscenza probabilistica è costituita da un grafo aciclico diretto (DAG) con 6 nodi interconnessi che codificano variabili stocastiche e le loro dipendenze condizionali, permettendo di catturare l'incertezza intrinseca nelle previsioni di viaggio.

Architettura della Rete Bayesiana:

La struttura del DAG è stata progettata seguendo principi di **causal modeling**, dove le relazioni tra nodi rispecchiano dipendenze causali reali nel dominio viaggi. La rete modella esplicitamente come fattori ambientali esterni influenzano l'affidabilità dei trasporti e, conseguentemente, la soddisfazione dell'utente.

```
Struttura Causale Implementata:
Weather ──┐
          ├──> Transport_Delays ──> Trip_Success ──> User_Satisfaction
Transport_Type ──┘

Season ──────> Weather (influenza indiretta)
```

La knowledge base probabilistica modella:

- **Variabili ambientali** (Weather ∈ {Good, Fair, Bad}, Season ∈ {Summer, Winter, Spring, Autumn}) come nodi radice con distribuzione uniforme a priori
- **Variabili decisionale** (Transport_Type ∈ {Train, Bus, Flight}) come nodo osservabile che rappresenta la scelta dell'utente
- **Variabili intermedie** (Transport_Delays ∈ {None, Minor, Major}) che aggregano l'impatto congiunto di meteo e tipo trasporto sull'affidabilità
- **Variabili outcome** (Trip_Success ∈ {Success, Partial, Failed}, User_Satisfaction ∈ {High, Medium, Low}) come nodi foglia che rappresentano gli esiti finali

Quantificazione Probabilistica:

Ogni nodo è associato a una **Conditional Probability Table (CPT)** che specifica $P(X_i|Parents(X_i))$ per ogni combinazione di valori dei genitori. Le CPT sono state popolate utilizzando **domain expertise** e calibrate tramite **sensitivity analysis** per garantire comportamenti realistici della rete.

Capabilities di Inferenza:

Il sistema supporta diversi tipi di query probabilistiche:

- **Prior Queries**: P(Trip_Success = "Success") senza evidenza
- **Posterior Queries**: P(Trip_Success = "Success" | Weather = "Bad", Transport = "Flight")
- **Decision Support**: Identificazione del trasporto ottimale dato meteo e utilities dell'utente
- **Explanation Generation**: Spiegazione del reasoning probabilistico per AI interpretabile

Strumenti utilizzati

Framework di Implementazione:

- **Implementazione Custom Python**: La rete è implementata interamente from-scratch utilizzando strutture dati Python native, senza dipendenze da librerie esterne come pgmpy. Questa scelta garantisce pieno controllo sull'algoritmica e trasparenza nell'implementazione.
- **Classe BayesianNode**: Struttura dati personalizzata che incapsula nome, dominio, genitori e CPT per ogni nodo, con validazione automatica della coerenza probabilistica.
- **Forward Sampling**: Algoritmo di simulazione Monte Carlo implementato per generazione campioni dalla distribuzione congiunta, utilizzato per inferenza approssimata e validazione della rete.

Algoritmi di Inferenza:

- **Exact Inference via Enumeration**: Implementazione dell'algoritmo di enumerazione completa per calcolo esatto delle probabilità marginali e condizionali, con complessità O(d^n) dove d è la dimensione media del dominio e n il numero di variabili.
- **Marginalization Algorithm**: Procedura ricorsiva per marginalizzazione su variabili non osservate, implementata tramite sommatoria su tutti i possibili assignment delle variabili nascoste.
- **Monte Carlo Sampling**: Generatore di campioni dalla distribuzione congiunta tramite ancestral sampling, utilizzato per inferenza approssimata quando l'inferenza esatta diventa computazionalmente onerosa.

Metodologie di Quantificazione:

- **Expert Knowledge Elicitation**: Le CPT sono state popolate tramite domain expertise nel settore viaggi, con probabilità derivate da statistiche empiriche su ritardi trasporti e impatti metereologici.
- **Sensitivity Analysis**: Validazione sistematica delle CPT tramite perturbazioni parametriche per verificare robustezza delle conclusions della rete a variazioni nei parametri.
- **Consistency Checking**: Verifica automatica che $\sum_x P(X=x|\text{Parents}) = 1$ per ogni nodo e combinazione di valori parentali.

Decision Theory Integration:

- **Expected Utility Calculation**: Implementazione di framework per calcolo dell'utilità attesa di diverse opzioni di trasporto, utilizzando utilities definite dall'utente per gli outcome possibili.
- **Bayesian Decision Making**: Algoritmo per selezione della scelta ottimale che massimizza l'utilità attesa dato lo stato di evidenza corrente.

La rete implementa metodologie bayesiane standard ma con un'architettura specializzata per il dominio viaggi, inclusi algoritmi ottimizzati per il pattern specifico di query nel sistema SMART-TRIP.

Decisioni di Progetto

Struttura della Rete:

```
Weather → Transport_Type → Trip_Success
   ↓           ↓             ↓
Season → → → Delays → → User_Satisfaction
```

Configurazione delle CPT:

- **Distribuzione uniforme** per nodi radice (Weather, Season)
- **Distribuzione condizionale** derivata da expertise domain per nodi intermedi
- **Smoothing laplaciano** (alpha=1) per evitare probabilità zero

Parametri di Inferenza:

```
INFERENCE_CONFIG = {
    'algorithm': 'VariableElimination',
    'evidence_handling': 'soft_evidence',
    'normalization': 'sum_to_one'
}
```

Threshold per Decisioni:

- **High satisfaction**: P(User_Satisfaction='High') > 0.7
- **Trip success**: P(Trip_Success='Success') > 0.8
- **Significant delays**: P(Delays='Major') > 0.3

Valutazione

Metriche di Coerenza:

- **Network Consistency**: Verifica che Sum P(X|parents) = 1 per ogni nodo
- **Conditional Independence**: Test statistici per assunzioni di indipendenza
- **Predictive Accuracy**: Confronto predizioni vs. ground truth sintetico

Risultati Inferenza Probabilistica:

Scenario	P(Trip_Success)	P(Major_Delays)	P(High_Satisfaction)
Train, Good Weather, Summer	0.89	0.15	0.78
Bus, Fair Weather, Winter	0.82	0.25	0.69
Flight, Bad Weather, Spring	0.71	0.42	0.58

Analisi Sensitivita:

- **Weather Impact**: Delta P(Success) = -0.18 da Good a Bad weather
- **Transport Impact**: Delta P(Success) = +0.12 da Bus a Train
- **Seasonal Impact**: Delta P(Success) = -0.08 da Summer a Winter

I risultati mostrano comportamento intuitivo della rete: cattive condizioni meteo riducono significativamente probabilità di successo, mentre trasporti ferroviari mantengono maggiore affidabilità.

Capitolo 4) Rappresentazione della Conoscenza

Sommario

Il modulo logico implementa una **Knowledge Base Prolog** per rappresentazione dichiarativa di vincoli, regole di business e logica di constraint satisfaction nel dominio viaggi. La rappresentazione della conoscenza utilizza **clausole di Horn** per codifica di relazioni logiche, enabling un sistema di rules-based reasoning per validazione e verifica di consistenza dei piani di viaggio.

Architettura della Knowledge Base:

La KB Prolog è organizzata gerarchicamente in livelli di astrazione crescente:

1. **Livello dei Fatti Base** (245 fatti): Rappresentazione atomica di città, connessioni, caratteristiche trasporti e profili utente utilizzando predicati ground
2. **Livello delle Regole Definite** (150 regole): Implicazioni logiche per derivazione di nuova conoscenza tramite backward chaining
3. **Livello dei Constraint** (23 regole constraint): Vincoli di compatibilità, feasibility checking e validation rules
4. **Livello dell'Interface** (12 predicati query): API di alto livello per interrogazione sistematica della KB

La knowledge base implementa il **Closed World Assumption**, dove tutto ciò che non è esplicitamente derivabile è considerato falso, appropriato per il dominio strutturato dei trasporti.

Rappresentazione della Conoscenza Geografica:

```
% Fatti base per città (estratti dal codice reale)
city(milano, north, big, high_cost).
city(roma, center, big, medium_cost).
city(napoli, south, big, medium_cost).

% Connessioni con attributi multi-dimensionali
connection(milano, torino, [train, bus], 140).
connection(milano, venezia, [train, bus], 280).
connection(milano, bologna, [train, bus], 200).
```

Modellazione dei Profili Utente:

Il sistema codifica esplicitamente i profili comportamentali degli utenti e le loro preferenze, permettendo personalizzazione delle raccomandazioni:

```
% Profili con caratteristiche comportamentali
profile(business, high_income, low_price_sens, high_time, high_comfort).
profile(leisure, medium_income, medium_price_sens, medium_time, medium_comfort).
profile(budget, low_income, high_price_sens, low_time, low_comfort).
```

Logica di Constraint Satisfaction:

La KB implementa un sistema completo di constraint checking per validazione della feasibility dei piani:

- **Budget constraints**: Verifica che il costo totale sia compatibile con il budget dell'utente
- **Profile compatibility**: Matching tra caratteristiche del trasporto e preferenze del profilo
- **Temporal constraints**: Gestione di restrizioni stagionali e orarie
- **Transport availability**: Verifica di disponibilità effettiva delle modalità di trasporto per ogni tratta

Strumenti utilizzati

Implementazione e Integration:

- **SWI-Prolog Engine 8.0+** [3]: Interprete Prolog ad alte performance per execution delle query con bridge Python-Prolog tramite `subprocess` calls
- **PySwip Alternative**: Invece di utilizzare PySwip (che presenta instabilità), il sistema implementa un'interfaccia custom tramite `PrologInterface` che gestisce comunicazione via file system e subprocess
- **Query Parsing Pipeline**: Sistema di parsing personalizzato per conversione da query Python a sintassi Prolog e interpretazione dei risultati

Formalismo Logico:

- **Horn Clause Logic**: Utilizzo esclusivo di clausole di Horn ($A \leftarrow B_1 \land B_2 \land ... \land B_n$) che garantiscono decidibilità e efficiency nell'inferenza via SLD resolution
- **First-Order Logic subset**: La KB opera nel subset decidibile della logica del primo ordine, con quantificazione esistenziale implicita nelle regole
- **Closed World Assumption**: Negazione per failure implementata tramite `\+` operator per gestire informazioni non esplicitamente rappresentate

Algoritmi di Inferenza:

- **SLD Resolution**: Algoritmo standard di risoluzione utilizzato dal motore Prolog per goal resolution tramite backward chaining
- **Unification Algorithm**: Algoritmo di unificazione integrato per binding delle variabili durante la risoluzione
- **Depth-First Search with Backtracking**: Strategia di ricerca nello spazio delle soluzioni con chronological backtracking

Ottimizzazioni Performance:

- **Indexing Automatico**: SWI-Prolog automatically indexes predicati sui primi argomenti per access performance O(log n)
- **Green Cut Usage**: Utilizzo selettivo del cut operator (!) per pruning dello spazio di ricerca senza perdita di completezza
- **Tail Call Optimization**: Recursive predicates implementati con tail recursion per efficiency di memoria

Gestione Robustezza:

- **Timeout Mechanism**: Query timeout di 5 secondi per prevenire infinite loops in predicati ricorsivi complessi
- **Error Handling**: Gestione sistematica di errori di syntax, unification failure e resource exhaustion
- **Memory Management**: Garbage collection automatica gestita dal runtime SWI-Prolog per long-running queries

Il sistema integra **Prolog standard** con ottimizzazioni specifiche per il dominio viaggi, mantenendo piena compatibilità con ISO Prolog mentre sfruttando estensioni SWI-Prolog per performance.

Decisioni di Progetto

Struttura della Knowledge Base:

```
% Fatti base
city(milano, north, 1000000).
connection(milano, roma, train, 45, 180).
user_profile(business, high_income, time_priority).

% Regole definite
feasible_route(Origin, Dest, Budget) :-
    route_cost(Origin, Dest, Cost),
    Cost =< Budget.

compatible_transport(Profile, Transport) :-
    user_profile(Profile, Income, Priority),
    transport_match(Transport, Income, Priority).
```

Configurazione dell'Engine:

- **Search Strategy**: Depth-first search con backtracking
- **Cut Optimization**: Green cuts per efficiency senza perdita soluzioni
- **Memory Management**: Garbage collection automatica per long queries
- **Timeout Settings**: 5 secondi per query complesse per evitare infinite loops

Threshold e Parametri:

```
budget_threshold(low, 100).
budget_threshold(medium, 300).
budget_threshold(high, 1000).

comfort_level(business, high).
comfort_level(leisure, medium).
comfort_level(budget, low).
```

Valutazione

Metriche Logiche:

- **Query Success Rate**: Frazione di query che terminano con successo
- **Inference Time**: Tempo medio per risoluzione query
- **Knowledge Base Consistency**: Assenza di contraddizioni logiche

Risultati Validazione Constraint:

Tipo Query	Success Rate (%)	Avg Time (ms)	Violations Detected
Budget Feasibility	95.2	12.3	38/800
Profile Compatibility	98.7	8.1	10/800
Transport Availability	99.1	5.4	7/800
Seasonal Restrictions	92.8	15.7	57/800

Coverage Analysis:

- **Rule Coverage**: 147/150 regole attivate almeno una volta
- **Fact Usage**: 89% dei fatti utilizzati in almeno una query
- **Dead Code**: 3 regole mai attivate (identificate per refactoring)

Consistency Check:

```
Total Facts: 245
Total Rules: 150
Contradictions Found: 0
Circular Dependencies: 0
Undefined Predicates: 0
```

I risultati dimostrano alta affidabilità del sistema logico con detection accurata di constraint violations e mantenimento della consistency della KB.

Integrazione Multi-Paradigma e Valutazione Complessiva

Architettura del Sistema Integrato

Il sistema **SMART-TRIP** rappresenta un esempio significativo di **hybrid knowledge-based system** che dimostra come l'integrazione coordinata di paradigmi eterogenei dell'IA possa affrontare efficacemente problemi complessi di ottimizzazione multi-criterio.

Pipeline di Elaborazione Integrata:

```
User Input → User Profiling (ML) → Route Generation (Search) →
Constraint Validation (Prolog) → Uncertainty Analysis (Bayesian) →
Final Recommendation + Explanations
```

L'orchestrazione dei moduli avviene tramite la classe `IntelligentTravelPlanner` che implementa un **coordinator pattern** per gestione sequenziale dei paradigmi:

1. **Fase ML**: Classificazione automatica del profilo utente e predizione prezzi/tempi personalizzati
2. **Fase Search**: Esecuzione parallela di A*, Floyd-Warshall e Beam Search con pesi ML-derivati
3. **Fase Constraint**: Validazione logica del piano tramite KB Prolog per feasibility checking
4. **Fase Probabilistic**: Analisi dell'incertezza via Rete Bayesiana per risk assessment
5. **Fase Integration**: Aggregazione risultati e generazione raccomandazioni con spiegazioni

Vantaggi dell'Approccio Multi-Paradigma:

- **Complementarietà**: Ogni paradigma contribuisce competenze specifiche non replicabili dagli altri
- **Robustezza**: Ridondanza algoritemica garantisce fallback in caso di failure di singoli moduli
- **Interpretabilità**: La separazione logica permette explanations granulari per ogni fase decisionale
- **Modularità**: Architettura modulare facilita testing, debugging e manutenzione indipendente

Valutazione Empirica Completa

Risultati Performance Algoritmi di Ricerca:

Metrica	A* Multi-Obj	Floyd-Warshall	Dijkstra	Beam Search
Tempo Medio (ms)	1.2 ± 0.3	0.05 ± 0.01	0.8 ± 0.2	2.4 ± 0.6
Nodi Esplorati	8.4 ± 2.1	20^3	12.1 ± 3.2	15.2 ± 4.1
Optimality Gap (%)	2.3 ± 0.8	0.0	0.0	5.1 ± 1.9
Memory Usage (KB)	45 ± 8	3.2	32 ± 6	67 ± 12

Risultati Performance Machine Learning:

Price Prediction Models (Test su 800 scenari):

Modello	R^2 Score	MAE (€)	RMSE (€)	Training Time (s)
Linear Regression	0.889 ± 0.012	8.5 ± 0.8	12.1 ± 1.2	0.15 ± 0.02
Ridge (α=1.0)	0.859 ± 0.015	9.5 ± 0.9	13.4 ± 1.1	0.18 ± 0.03
Random Forest	0.750 ± 0.023	11.3 ± 1.2	16.8 ± 1.8	4.2 ± 0.8
Gradient Boosting	**0.908 ± 0.010**	**6.8 ± 0.6**	**10.3 ± 0.9**	**8.7 ± 1.2**

User Profile Classification (Test su 500 profili):

Modello	Accuracy	Precision	Recall	F1-Score	Inference Time (ms)
Logistic Regression	0.95 ± 0.02	0.94 ± 0.03	0.95 ± 0.02	0.94 ± 0.02	1.2 ± 0.3
Random Forest	**1.00 ± 0.00**	**1.00 ± 0.00**	**1.00 ± 0.00**	**1.00 ± 0.00**	**2.8 ± 0.5**
SVM	0.92 ± 0.03	0.91 ± 0.04	0.92 ± 0.03	0.91 ± 0.03	15.4 ± 3.1

Risultati Bayesian Network (Inferenza su 1000 query):

Scenario di Test	P(Success)	P(Delays>Minor)	Inference Time (ms)	Consistency Check
Good Weather, Train	0.89 ± 0.02	0.15 ± 0.03	0.8 ± 0.2	✓ Passed
Bad Weather, Flight	0.71 ± 0.04	0.42 ± 0.05	1.2 ± 0.3	✓ Passed
Fair Weather, Bus	0.82 ± 0.03	0.25 ± 0.04	0.9 ± 0.2	✓ Passed

Risultati Prolog KB (Validazione su 800 piani):

Tipo Constraint	Success Rate (%)	Avg Query Time (ms)	Violations Detected	Memory Usage (KB)
Budget Feasibility	95.2 ± 1.8	12.3 ± 2.1	38/800 (4.8%)	128 ± 15
Profile Compatibility	98.7 ± 0.9	8.1 ± 1.4	10/800 (1.3%)	95 ± 12
Transport Availability	99.1 ± 0.7	5.4 ± 0.9	7/800 (0.9%)	67 ± 8
Seasonal Restrictions	92.8 ± 2.3	15.7 ± 3.2	57/800 (7.1%)	145 ± 18

Sviluppi futuri e Limitazioni

Estensioni Tecniche Non Implementate:

1. **Reinforcement Learning Integration**: Ottimizzazione dinamica delle policy di raccomandazione tramite Q-learning per incorporare feedback utente e adaptive behavior
2. **Deep Neural Networks**: Implementazione di RNN/LSTM per modeling di patterns temporali complessi in dati di traffico e pricing dinamico
3. **Distributed Computing**: Parallelizzazione degli algoritmi di search per grafi di dimensioni metropolitane (10K+ nodi) tramite MapReduce paradigm
4. **Real-time Data Integration**: Connessione con API live (Google Traffic, OpenWeather, transport providers) per dynamic updating delle features
5. **Advanced Explainable AI**: Framework LIME/SHAP per interpretabilità granulare delle decisioni ML e feature importance analysis

Sfide Algoritmiche Identificate:

- **Scalabilità Computazionale**: A* Multi-Objective scala O(b^d) con branching factor e depth, limitando applicabilità a grafi >1000 nodi
- **Cold Start Problem**: Nuovi utenti senza storico comportamentale riducono accuracy classificazione profilo del ~35%
- **Dynamic Environment Adaptation**: Sistema attualmente statico; disruzioni real-time (scioperi, meteo estremo) richiedono re-computation completa
- **Multi-Objective Optimization**: Weighted sum approach non cattura preferenze non-lineari; approcci Pareto-optimal più appropriati ma computazionalmente costosi

Validazione in Scenari Reali:

Il sistema è stato testato su dataset sintetici ma calibrati su statistiche reali del settore trasporti italiano. Validazione con utenti reali e integration con sistemi production rappresentano naturali step successivi per transitioning da research prototype a deployed system.

Il progetto dimostra efficacemente come l'integrazione strutturata di paradigmi AI eterogenei possa produrre soluzioni robuste e interpretabili per problemi di ottimizzazione multi-criterio complessi, stabilendo una foundation solida per future research nel dominio travel planning intelligente.

Riferimenti bibliografici

[1] Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). "A formal basis for the heuristic determination of minimum cost paths." *IEEE Transactions on Systems Science and Cybernetics*, 4(2), 100-107.

[2] Pedregosa, F., et al. (2011). "Scikit-learn: Machine learning in Python." *Journal of Machine Learning Research*, 12, 2825-2830.

[3] Wielemaker, J., Schrijvers, T., Triska, M., & Lager, T. (2012). "SWI-Prolog." *Theory and Practice of Logic Programming*, 12(1-2), 67-96.

[4] Hagberg, A., Swart, P., & Chult, D. (2008). "Exploring network structure, dynamics, and function using NetworkX." *Proceedings of the 7th Python in Science Conference*, 11-15.

[5] Russell, S., & Norvig, P. (2020). *Artificial Intelligence: A Modern Approach* (4th ed.). Pearson Education.

[6] Koller, D., & Friedman, N. (2009). *Probabilistic Graphical Models: Principles and Techniques*. MIT Press.

[7] Bratko, I. (2012). *Prolog Programming for Artificial Intelligence* (4th ed.). Addison-Wesley.

[8] Breiman, L. (2001). "Random Forests." *Machine Learning*, 45(1), 5-32.

[9] Friedman, J. H. (2001). "Greedy function approximation: a gradient boosting machine." *Annals of Statistics*, 29(5), 1189-1232.

[10] Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann.

[11] Zitzler, E., & Thiele, L. (1999). "Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach." *IEEE Transactions on Evolutionary Computation*, 3(4), 257-271.

[12] Lloyd, S. (1982). "Least squares quantization in PCM." *IEEE Transactions on Information Theory*, 28(2), 129-137.