PyVRP 統一API設計仕様書

1. API設計概要

データモデル基盤

本APIは**Pydantic**を使用した堅牢なデータ検証システムを採用しています。すべてのリクエスト・レスポンスモデルはPydanticの`BaseModel`を継承し、自動的な型検証、データ変換、API文書生成を提供します。

```
from pydantic import BaseModel, Field, validator
from typing import List, Optional, Union
```

エンドポイント

```
POST /api/pyvrp/solve
Content-Type: application/json
```

リクエスト構造

```
{
  "clients": [...],
  "depots": [...],
  "vehicle_types": [...],
  "distance_matrix": [...],  // オプション
  "duration_matrix": [...],  // オプション
  "max_runtime": 60
}
```

2. Pydanticデータモデル詳細

ClientModel (Pydantic BaseModel)

```
class ClientModel(BaseModel):
    """統合クライアントモデル - 全VRP亜種対応"""
    x: int = Field(description="X座標（整数）")
    y: int = Field(description="Y座標（整数）")
    delivery: Union[int, List[int]] = Field(default=0, description="配送需要
（整数または整数リスト）")
    pickup: Union[int, List[int]] = Field(default=0, description="集荷需要
（整数または整数リスト）")
    service_duration: int = Field(default=10, description="サービス時間
（分）")
    tw_early: Optional[int] = Field(default=0, description="時間窓開始（真夜中
```

```
からの分）")
    tw_late: Optional[int] = Field(default=1440, description="時間窓終了（真
夜中からの分）")
    release_time: Optional[int] = Field(default=0, description="リリース時
刻")
    prize: Optional[int] = Field(default=0, description="訪問時の報酬（PC-
VRP）")
    required: bool = Field(default=True, description="必須訪問フラグ")

    @validator('tw_late')
    def validate_time_window(cls, v, values):
        if 'tw_early' in values and v < values['tw_early']:
            raise ValueError('tw_late must be greater than or equal to
tw_early')
        return v

# JSON例:
{
    "x": 100,                    # X座標（整数）
    "y": 200,                    # Y座標（整数）
    "delivery": 50,              # 配送需要（整数または整数リスト）
    "pickup": 0,                 # 集荷需要（整数または整数リスト）
    "service_duration": 10,      # サービス時間（分）
    "tw_early": 480,             # 時間窓開始（8:00 = 480分）
    "tw_late": 1020,             # 時間窓終了（17:00 = 1020分）
    "release_time": 0,           # リリース時刻
    "prize": 100,                # 訪問時の報酬
    "required": true             # 必須訪問フラグ
}
```

DepotModel (Pydantic BaseModel)

```
class DepotModel(BaseModel):
    """統合デポモデル"""
    x: int = Field(description="X座標（整数）")
    y: int = Field(description="Y座標（整数）")

# JSON例:
{
    "x": 0,                      # X座標（整数）
    "y": 0                       # Y座標（整数）
}
```

VehicleTypeModel (Pydantic BaseModel)

```
class VehicleTypeModel(BaseModel):
    """統合車両タイプモデル"""
    num_available: int = Field(description="利用可能台数")
    capacity: Union[int, List[int]] = Field(description="積載容量（整数または
```

```
整数リスト）")
    start_depot: int = Field(description="出発デポインデックス")
    end_depot: Optional[int] = Field(default=None, description="到着デポイン
デックス（Noneで出発点に戻る）")
    fixed_cost: int = Field(default=0, description="固定費")
    tw_early: Optional[int] = Field(default=0, description="シフト開始時刻
（分）")
    tw_late: Optional[int] = Field(default=1440, description="シフト終了時刻
（分）")
    max_duration: Optional[int] = Field(default=480, description="最大ルート
時間（分）")
    max_distance: Optional[int] = Field(default=200000, description="最大ル
ート距離（メートル）")

    @validator('end_depot')
    def validate_end_depot(cls, v, values):
        if v is None:
            return values.get('start_depot', 0)
        return v

# JSON例:
{
    "num_available": 5,           # 利用可能台数
    "capacity": 1000,             # 積載容量（整数または整数リスト）
    "start_depot": 0,             # 出発デポインデックス
    "end_depot": 0,               # 到着デポインデックス（null可）
    "fixed_cost": 100,            # 固定費
    "tw_early": 0,                # シフト開始時刻
    "tw_late": 1440,              # シフト終了時刻（24時間）
    "max_duration": 480,          # 最大ルート時間（分）
    "max_distance": 200000        # 最大ルート距離（メートル）
}
```

3. pandas DataFrame → JSON 変換詳細（Pydantic検証付き）

3.1 基本的な変換フロー

```
import pandas as pd
import numpy as np
import json
from pydantic import ValidationError
from app.models.vrp_unified_models import VRPProblemData, ClientModel,
DepotModel, VehicleTypeModel

def dataframes_to_vrp_json(
    locations_df: pd.DataFrame,
    vehicle_types_df: pd.DataFrame = None,
    time_windows_df: pd.DataFrame = None,
    depot_indices: list = None
) -> dict:
    """
```

```
    pandas DataFrameからPyVRP API用のJSONデータを生成
    Pydanticモデルによる自動検証付き
    """

    # デフォルトのデポインデックス
    if depot_indices is None:
        depot_indices = [0]

    # 1. デポとクライアントの分離
    depots = []
    clients = []

    for idx, row in locations_df.iterrows():
        if idx in depot_indices:
            depots.append({
                "x": int(row.get('x', row.get('lon', 0) * 10000)),
                "y": int(row.get('y', row.get('lat', 0) * 10000))
            })
        else:
            clients.append(_create_client(row, idx, time_windows_df))

    # 2. 車両タイプの生成
    vehicle_types = _create_vehicle_types(vehicle_types_df, depot_indices)

    # 3. 距離行列の計算（オプション）
    distance_matrix = _calculate_distance_matrix(locations_df)

    # 4. Pydanticモデルによる検証
    try:
        vrp_data = VRPProblemData(
            clients=clients,
            depots=depots,
            vehicle_types=vehicle_types,
            distance_matrix=distance_matrix,
            max_runtime=60
        )
        return vrp_data.dict()
    except ValidationError as e:
        print(f"データ検証エラー: {e}")
        raise
```

3.2 クライアントデータの変換（Pydantic検証付き）

```
def _create_client(row: pd.Series, idx: int, time_windows_df: pd.DataFrame
= None) -> ClientModel:
    """
    DataFrameの1行からPydantic ClientModelを生成
    自動的な型検証とデータ変換を実行
    """
    client = {
        # 座標（緯度経度の場合は10000倍して整数化）
```

```
            "x": int(row.get('x', row.get('lon', 0) * 10000)),
            "y": int(row.get('y', row.get('lat', 0) * 10000)),

            # 需要（デフォルト0）
            "delivery": int(row.get('demand', row.get('delivery', 0))),
            "pickup": int(row.get('pickup', 0)),

            # サービス時間（分単位、デフォルト10分）
            "service_duration": int(row.get('service_time',
row.get('service_duration', 10))),

            # 必須訪問（デフォルトTrue）
            "required": bool(row.get('required', True)),

            # 賞金（PC-VRP用、デフォルト0）
            "prize": int(row.get('prize', 0))
        }

        # 時間窓の設定
        if time_windows_df is not None:
            tw_row = time_windows_df[time_windows_df['location_id'] == idx]
            if not tw_row.empty:
                client["tw_early"] = int(tw_row.iloc[0]['tw_early'] * 60)  # 時
間→分
                client["tw_late"] = int(tw_row.iloc[0]['tw_late'] * 60)
        else:
            # デフォルト時間窓（営業時間全体）
            client["tw_early"] = 0
            client["tw_late"] = 1440  # 24時間

        # Pydanticモデルで検証
        try:
            return ClientModel(**client)
        except ValidationError as e:
            print(f"クライアントデータ検証エラー (ID: {idx}): {e}")
            raise
```

3.3 車両タイプの変換（Pydantic検証付き）

```
def _create_vehicle_types(vehicle_types_df: pd.DataFrame, depot_indices:
list) -> List[VehicleTypeModel]:
    """
    車両タイプDataFrameからPydantic VehicleTypeModelリストを生成
    自動的な型検証とデータ変換を実行
    """
    if vehicle_types_df is None or vehicle_types_df.empty:
        # デフォルト車両タイプ（Pydantic検証付き）
        try:
            default_vt = VehicleTypeModel(
                num_available=10,
                capacity=1000,
```

```
                start_depot=0,
                end_depot=0,
                fixed_cost=0,
                tw_early=0,
                tw_late=1440,
                max_duration=480,
                max_distance=200000
            )
            return [default_vt]
        except ValidationError as e:
            print(f"デフォルト車両タイプ検証エラー: {e}")
            raise

    vehicle_types = []
    for _, row in vehicle_types_df.iterrows():
        vt = {
            "num_available": int(row.get('num_available', 1)),
            "capacity": int(row.get('capacity', 1000)),
            "start_depot": int(row.get('start_depot', depot_indices[0])),
            "fixed_cost": int(row.get('fixed_cost', 0)),
            "tw_early": int(row.get('shift_start', 0) * 60),
            "tw_late": int(row.get('shift_end', 24) * 60)
        }

        # オプション項目
        if 'end_depot' in row:
            vt["end_depot"] = int(row['end_depot'])
        if 'max_duration' in row:
            vt["max_duration"] = int(row['max_duration'] * 60)
        if 'max_distance' in row:
            vt["max_distance"] = int(row['max_distance'] * 1000)

        # Pydanticモデルで検証
        try:
            vehicle_types.append(VehicleTypeModel(**vt))
        except ValidationError as e:
            print(f"車両タイプデータ検証エラー: {e}")
            raise

    return vehicle_types
```

3.4 実用的な変換例

例1: 基本的なCVRP

```
# 位置データ
locations_df = pd.DataFrame({
    'name': ['Depot', 'Customer1', 'Customer2', 'Customer3'],
    'lat': [35.6762, 35.6854, 35.6586, 35.6908],
    'lon': [139.6503, 139.7531, 139.7454, 139.6909],
    'demand': [0, 20, 30, 25]
```

```
})

# 車両データ
vehicles_df = pd.DataFrame({
    'capacity': [100],
    'num_available': [3]
})

# JSON変換
vrp_json = dataframes_to_vrp_json(
    locations_df=locations_df,
    vehicle_types_df=vehicles_df,
    depot_indices=[0]
)
```

例2: 時間窓付きVRPTW

```
# 時間窓データ（時間単位）
time_windows_df = pd.DataFrame({
    'location_id': [1, 2, 3],
    'tw_early': [8, 9, 10],    # 8:00, 9:00, 10:00
    'tw_late': [12, 15, 17]    # 12:00, 15:00, 17:00
})

# サービス時間を含む位置データ
locations_df['service_time'] = [0, 30, 45, 20]  # 分単位

vrp_json = dataframes_to_vrp_json(
    locations_df=locations_df,
    vehicle_types_df=vehicles_df,
    time_windows_df=time_windows_df,
    depot_indices=[0]
)
```

例3: マルチデポMDVRP

```
# 複数デポを含む位置データ
locations_df = pd.DataFrame({
    'name': ['Depot1', 'Depot2', 'Customer1', 'Customer2', 'Customer3'],
    'lat': [35.6762, 35.6894, 35.6854, 35.6586, 35.6908],
    'lon': [139.6503, 139.7742, 139.7531, 139.7454, 139.6909],
    'demand': [0, 0, 20, 30, 25]
})

# デポごとの車両設定
vehicles_df = pd.DataFrame({
    'capacity': [100, 150],
    'num_available': [2, 3],
```

```
    'start_depot': [0, 1]
})

vrp_json = dataframes_to_vrp_json(
    locations_df=locations_df,
    vehicle_types_df=vehicles_df,
    depot_indices=[0, 1]  # 最初の2つがデポ
)
```

3.5 距離行列の計算

```
def _calculate_distance_matrix(locations_df: pd.DataFrame) -> list:
    """
    座標から距離行列を計算（メートル単位の整数）
    """
    n = len(locations_df)
    matrix = []

    for i in range(n):
        row = []
        for j in range(n):
            if i == j:
                row.append(0)
            else:
                # ユークリッド距離または実際の道路距離
                dist = _euclidean_distance(
                    locations_df.iloc[i]['x'], locations_df.iloc[i]['y'],
                    locations_df.iloc[j]['x'], locations_df.iloc[j]['y']
                )
                row.append(int(dist))
        matrix.append(row)

    return matrix
```

4. APIレスポンス（Pydanticモデル）

UnifiedVRPSolution (Pydantic BaseModel)

```
class UnifiedVRPSolution(BaseModel):
    """統合VRPソリューションモデル"""
    status: str = Field(description="ソリューション状態: optimal, feasible,
infeasible, error")
    objective_value: float = Field(description="目的関数値")
    routes: List[UnifiedRouteModel] = Field(description="ルートリスト")
    computation_time: float = Field(description="計算時間（秒）")
    solver: str = Field(default="PyVRP", description="使用ソルバー")

class UnifiedRouteModel(BaseModel):
    """統合ルートモデル"""
```

```
    vehicle_type: int = Field(description="使用車両タイプインデックス")
    depot: int = Field(description="デポインデックス")
    clients: List[int] = Field(description="訪問クライアントインデックス順序")
    distance: int = Field(description="総ルート距離（メートル）")
    duration: int = Field(description="総ルート時間（分）")
    demand_served: Union[int, List[int]] = Field(description="総配送需要")

# JSON例:
{
    "status": "optimal",
    "objective_value": 12345,
    "routes": [
        {
            "vehicle_type": 0,
            "depot": 0,
            "clients": [2, 4, 1],
            "distance": 4567,
            "duration": 234,
            "demand_served": 75
        }
    ],
    "computation_time": 1.23,
    "solver": "PyVRP"
}
```

5. 完全な使用例（Pydantic検証付き）

```
import requests
import pandas as pd
from pydantic import ValidationError
from app.models.vrp_unified_models import VRPProblemData,
UnifiedVRPSolution

# 1. データ準備
customers_df = pd.read_csv('customers.csv')
vehicles_df = pd.read_csv('vehicles.csv')

# 2. JSON変換（Pydantic検証付き）
try:
    vrp_data = dataframes_to_vrp_json(
        locations_df=customers_df,
        vehicle_types_df=vehicles_df,
        depot_indices=[0]
    )
    print("✅ データ検証成功")
except ValidationError as e:
    print(f"❌ データ検証エラー: {e}")
    exit(1)

# 3. API呼び出し
response = requests.post(
```

```
    'http://localhost:8000/api/pyvrp/solve',
    json=vrp_data
)

# 4. 結果処理（Pydanticレスポンス検証付き）
if response.status_code == 200:
    try:
        # レスポンスをPydanticモデルで検証
        solution = UnifiedVRPSolution(**response.json())
        print("✅ レスポンス検証成功")

        for route in solution.routes:
            print(f"Route: Depot -> {' -> '.join(map(str, route.clients))}
-> Depot")
            print(f"Distance: {route.distance}m")
    except ValidationError as e:
        print(f"❌ レスポンス検証エラー: {e}")
else:
    print(f"❌ API呼び出しエラー: {response.status_code}")
```

6. Pydantic利用の利点

6.1 自動データ検証

- **型安全性**: 整数、文字列、リストなどの型が自動検証される
- **範囲チェック**: 座標、時間窓、容量などの妥当性が自動確認される
- **必須フィールド**: 必要なデータの欠損を事前に検出

6.2 エラーハンドリング

```
try:
    client = ClientModel(x=100, y=200, tw_late=480, tw_early=600)  # エラー
例
except ValidationError as e:
    print(e.json())  # 詳細なエラー情報
```

6.3 自動API文書生成

FastAPIと組み合わせることで、Pydanticモデルから自動的にOpenAPI/Swagger文書が生成される

6.4 IDE支援

型ヒントにより、IDEでの自動補完とエラー検出が向上

この設計により、pandas DataFrameから直接PyVRP APIを呼び出すことが可能になり、**Pydanticによる堅牢なデータ検証**を通じて、データサイエンティストやAIエージェントが安全かつ容易にVRP問題を解決できるようになります。