

Nutrient Mobility Model - Crop Physiologist Expert Guide

Physiological Foundation: The Plant's Internal Transport Network

Understanding Nutrient Mobility: The Vascular Highway System

As a crop physiologist, nutrient mobility represents one of the most elegant examples of plant resource management. Plants have evolved sophisticated vascular systems - the xylem and phloem - that function as biological highways, constantly redistributing nutrients to meet changing demands throughout the plant's life cycle.

The Evolutionary Logic of Nutrient Mobility

From an evolutionary perspective, nutrient mobility allows plants to:

1. **Optimize Resource Allocation:** Move nutrients from low-demand to high-demand tissues
2. **Respond to Environmental Changes:** Redistribute resources when conditions change
3. **Maximize Reproductive Success:** Channel nutrients to developing seeds and fruits
4. **Survive Stress Periods:** Mobilize stored nutrients during adverse conditions
5. **Extend Tissue Lifespan:** Recycle nutrients from aging tissues before they die

The Two-Highway System: Xylem and Phloem

XYLEM: The Water Highway

- **Direction:** Unidirectional (roots → shoots)
- **Driving Force:** Transpiration-induced tension
- **Transport Speed:** Fast (up to 40 m/hour in trees)
- **Cargo:** Water, dissolved minerals, some organic compounds
- **pH:** Slightly acidic to neutral (pH 5-7)

PHLOEM: The Sugar Highway

- **Direction:** Bidirectional (source → sink)
- **Driving Force:** Pressure flow (osmotic pressure gradients)
- **Transport Speed:** Moderate (0.3-1.5 m/hour)
- **Cargo:** Sugars, amino acids, proteins, hormones, minerals
- **pH:** Alkaline (pH 7.5-8.5)

Mobile Nutrients: The Frequent Travelers

python

```
def analyze_mobile_nutrient_transport(nutrient_type, source_concentration,
                                     sink_demand, transport_conditions):
```

```
    """
```

```
    Model transport of mobile nutrients (N, P, K, Mg, S)
```

```
    Mobile nutrients can be readily redistributed because:
```

1. They're not structurally bound to cell walls
2. They have specific transport proteins in phloem
3. They can be remobilized from storage tissues
4. Their transport forms are phloem-compatible

```
    """
```

```
mobile_nutrient_properties = {
```

```
    'nitrogen': {
```

```
        'transport_forms': ['glutamine', 'asparagine', 'arginine', 'nitrate'],
```

```
        'phloem_concentration_range': (10, 50), # mM
```

```
        'remobilization_efficiency': 0.80,
```

```
        'transport_speed_factor': 1.0,
```

```
        'storage_locations': ['protein_bodies', 'vacuoles', 'chloroplasts'],
```

```
        'biochemical_function': 'proteins_enzymes_chlorophyll'
```

```
    },
```

```
    'phosphorus': {
```

```
        'transport_forms': ['phosphate', 'phosphate_esters', 'phytic_acid'],
```

```
        'phloem_concentration_range': (5, 25), # mM
```

```
        'remobilization_efficiency': 0.70,
```

```
        'transport_speed_factor': 0.8,
```

```
        'storage_locations': ['vacuoles', 'protein_bodies'],
```

```
        'biochemical_function': 'ATP_nucleic_acids_membranes'
```

```
    },
```

```
    'potassium': {
```

```
        'transport_forms': ['K_ion'],
```

```
        'phloem_concentration_range': (80, 150), # mM
```

```
        'remobilization_efficiency': 0.90,
```

```
        'transport_speed_factor': 1.2,
```

```
        'storage_locations': ['vacuoles', 'cytoplasm'],
```

```
        'biochemical_function': 'enzyme_activation_osmoregulation'
```

```
    },
```

```
    'magnesium': {
```

```
        'transport_forms': ['Mg_ion', 'Mg_chelates'],
```

```
        'phloem_concentration_range': (3, 15), # mM
```

```
        'remobilization_efficiency': 0.60,
```

```
        'transport_speed_factor': 0.9,
```

```
        'storage_locations': ['chloroplasts', 'vacuoles'],
```

```

        'biochemical_function': 'chlorophyll_enzyme_cofactor'
    },
    'sulfur': {
        'transport_forms': ['sulfate', 'glutathione', 'methionine'],
        'phloem_concentration_range': (2, 12), # mM
        'remobilization_efficiency': 0.65,
        'transport_speed_factor': 0.7,
        'storage_locations': ['vacuoles', 'protein_bodies'],
        'biochemical_function': 'amino_acids_proteins'
    }
}

if nutrient_type not in mobile_nutrient_properties:
    return None

props = mobile_nutrient_properties[nutrient_type]

# Calculate transport capacity
base_transport_rate = calculate_base_transport_rate(
    source_concentration, props['phloem_concentration_range']
)

# Environmental modifiers
temperature_factor = calculate_temperature_effect_on_transport(
    transport_conditions.get('temperature', 22)
)

water_status_factor = calculate_water_status_effect_on_transport(
    transport_conditions.get('water_potential', -0.5)
)

# Sink demand factor (stronger sinks pull more nutrients)
sink_strength_factor = calculate_sink_strength_effect(sink_demand)

# Actual transport rate
transport_rate = (
    base_transport_rate *
    props['transport_speed_factor'] *
    temperature_factor *
    water_status_factor *
    sink_strength_factor
)

# Calculate remobilization potential

```

```

remobilization_potential = (
    source_concentration *
    props['remobilization_efficiency'] *
    temperature_factor
)

return {
    'nutrient': nutrient_type,
    'transport_rate': transport_rate,
    'remobilization_potential': remobilization_potential,
    'transport_forms': props['transport_forms'],
    'limiting_factors': identify_transport_limitations(transport_conditions),
    'efficiency_score': calculate_transport_efficiency(transport_rate, sink_demand)
}

def calculate_base_transport_rate(source_concentration, phloem_range):
    """
    Calculate base transport rate based on source concentration and phloem capacity

    Transport follows modified Michaelis-Menten kinetics:
    - Loading into phloem is saturable
    - Higher source concentration increases driving force
    - Phloem has finite carrying capacity
    """

    min_phloem_conc, max_phloem_conc = phloem_range

    # Saturable loading function
    if source_concentration <= 0:
        return 0

    # Michaelis-Menten-like loading
    km_loading = min_phloem_conc # Half-saturation for phloem loading
    max_loading_rate = max_phloem_conc

    loading_rate = (max_loading_rate * source_concentration) / (km_loading + source_concentration)

    return loading_rate

```

Immobile Nutrients: The Stay-at-Home Elements

python

```
def analyze_immobile_nutrient_characteristics(nutrient_type, tissue_age, stress_factors):
```

```
    """
```

```
    Model behavior of immobile nutrients (Ca, Fe, Mn, B, Cu, Zn, Mo)
```

```
    Immobile nutrients are difficult to redistribute because:
```

1. They're bound to structural components (cell walls, membranes)
2. They form stable complexes difficult to break down
3. They lack efficient phloem transport mechanisms
4. Their chemical forms are incompatible with phloem transport

```
    """
```

```
    immobile_nutrient_properties = {
```

```
        'calcium': {
```

```
            'primary_binding_sites': ['cell_walls', 'middle_lamella', 'membranes'],
```

```
            'transport_limitation': 'structural_binding',
```

```
            'mobility_index': 0.05, # Very low mobility
```

```
            'deficiency_symptoms': 'apical_necrosis_tip_burn',
```

```
            'redistribution_potential': 0.10,
```

```
            'stress_response': 'increased_binding_under_stress'
```

```
        },
```

```
        'iron': {
```

```
            'primary_binding_sites': ['chloroplasts', 'enzyme_complexes', 'ferritin'],
```

```
            'transport_limitation': 'oxidation_precipitation',
```

```
            'mobility_index': 0.15,
```

```
            'deficiency_symptoms': 'interveinal_chlorosis',
```

```
            'redistribution_potential': 0.20,
```

```
            'stress_response': 'increased_binding_to_prevent_toxicity'
```

```
        },
```

```
        'manganese': {
```

```
            'primary_binding_sites': ['chloroplasts', 'enzyme_cofactors'],
```

```
            'transport_limitation': 'enzyme_binding',
```

```
            'mobility_index': 0.25,
```

```
            'deficiency_symptoms': 'interveinal_chlorosis_necrotic_spots',
```

```
            'redistribution_potential': 0.30,
```

```
            'stress_response': 'competitive_binding_with_other_metals'
```

```
        },
```

```
        'boron': {
```

```
            'primary_binding_sites': ['cell_walls', 'membrane_complexes'],
```

```
            'transport_limitation': 'structural_integration',
```

```
            'mobility_index': 0.10,
```

```
            'deficiency_symptoms': 'growing_point_death',
```

```
            'redistribution_potential': 0.15,
```

```
            'stress_response': 'increased_demand_for_cell_wall_synthesis'
```

```

    }
}

if nutrient_type not in immobile_nutrient_properties:
    return None

props = immobile_nutrient_properties[nutrient_type]

# Age effect on mobility (older tissues may release some bound nutrients)
age_factor = calculate_age_effect_on_immobile_nutrients(tissue_age, nutrient_type)

# Stress can sometimes increase mobility through tissue breakdown
stress_factor = calculate_stress_effect_on_immobile_nutrients(stress_factors, nutrient_type)

# Actual mobility (very limited)
effective_mobility = props['mobility_index'] * age_factor * stress_factor

# Redistribution potential under extreme conditions
redistribution_potential = props['redistribution_potential'] * stress_factor

return {
    'nutrient': nutrient_type,
    'base_mobility': props['mobility_index'],
    'effective_mobility': effective_mobility,
    'redistribution_potential': redistribution_potential,
    'binding_sites': props['primary_binding_sites'],
    'transport_limitation': props['transport_limitation'],
    'management_implications': generate_immobile_nutrient_management_advice(nutrient_type, props)
}

def calculate_age_effect_on_immobile_nutrients(tissue_age_days, nutrient_type):
    """
    Older tissues may release some immobile nutrients through natural breakdown
    """

    # Age effects vary by nutrient
    age_responses = {
        'calcium': 0.1, # Minimal release even from old tissues
        'iron': 0.3, # Some release from senescing chloroplasts
        'manganese': 0.4, # Moderate release from enzyme turnover
        'boron': 0.2 # Limited release from cell wall turnover
    }

    base_factor = age_responses.get(nutrient_type, 0.2)

```

```
# Sigmoid increase with age (most effect after 30 days)
```

```
if tissue_age_days > 30:
```

```
    age_effect = 1.0 + base_factor * (1.0 / (1.0 + math.exp(-0.1 * (tissue_age_days - 50))))
```

```
else:
```

```
    age_effect = 1.0
```

```
return age_effect
```

Transport Mechanisms and Driving Forces

Xylem Transport: The Transpiration Stream

```
python
```



```

def model_xylem_transport_dynamics(root_uptake_rate, transpiration_rate,
                                   stem_hydraulic_conductivity, nutrient_properties):
    """
    Model nutrient transport in the xylem transpiration stream

    Xylem transport is driven by:
    1. Transpiration-induced tension (primary driving force)
    2. Root pressure (minor contribution, mainly at night)
    3. Hydraulic conductivity of vascular system
    4. Nutrient solubility and mobility in xylem sap

    Key insight: Nutrients "ride along" with water movement
    """

    # Xylem sap composition and flow characteristics
    xylem_sap_properties = {
        'flow_rate': calculate_xylem_flow_rate(transpiration_rate, stem_hydraulic_conductivity),
        'ph': 6.2, # Slightly acidic
        'osmotic_potential': -0.3, # MPa
        'typical_concentrations': { # mM
            'nitrate': 15,
            'potassium': 8,
            'calcium': 12,
            'magnesium': 4,
            'phosphate': 2,
            'sulfate': 3
        }
    }

    # Calculate nutrient transport rate in xylem
    xylem_flow_rate = xylem_sap_properties['flow_rate'] # mL/hour

    # Nutrient-specific transport calculations
    transport_results = {}

    for nutrient, uptake_rate in root_uptake_rate.items():
        # Concentration in xylem sap
        if xylem_flow_rate > 0:
            xylem_concentration = uptake_rate / xylem_flow_rate # mM
        else:
            xylem_concentration = 0

        # Transport rate to shoots

```

```
transport_rate = xylem_concentration * xylem_flow_rate
```

```
# Nutrient-specific modifications
```

```
transport_efficiency = calculate_xylem_transport_efficiency(  
    nutrient, xylem_concentration, xylem_sap_properties  
)
```

```
actual_transport_rate = transport_rate * transport_efficiency
```

```
transport_results[nutrient] = {  
    'xylem_concentration': xylem_concentration,  
    'transport_rate': actual_transport_rate,  
    'transport_efficiency': transport_efficiency,  
    'distribution_pattern': predict_xylem_distribution_pattern(nutrient)  
}
```

```
return {  
    'xylem_flow_characteristics': xylem_sap_properties,  
    'nutrient_transport': transport_results,  
    'limiting_factors': identify_xylem_transport_limitations(transpiration_rate, stem_hydraulic_conductivity),  
    'temporal_patterns': model_diurnal_xylem_transport_patterns()  
}
```

```
def calculate_xylem_flow_rate(transpiration_rate, hydraulic_conductivity):
```

```
    """
```

```
    Calculate xylem flow rate from transpiration and hydraulic properties
```

```
    Flow rate ∝ Transpiration rate × Hydraulic conductivity
```

```
    """
```

```
# Simplified relationship (actual system is more complex)
```

```
# Typical values: transpiration 2-6 mmol H2O m-2 s-1
```

```
# Hydraulic conductivity varies by species and conditions
```

```
base_flow_rate = transpiration_rate * hydraulic_conductivity * 0.1 # Scaling factor
```

```
return max(0, base_flow_rate) # mL/hour per unit leaf area
```

```
def calculate_xylem_transport_efficiency(nutrient, concentration, xylem_properties):
```

```
    """
```

```
    Calculate transport efficiency for specific nutrients in xylem
```

```
    Factors affecting efficiency:
```

```
    1. Nutrient solubility at xylem pH
```

2. Precipitation/complex formation
3. Binding to xylem vessel walls
4. Chemical interactions with other ions

"""

```
efficiency_factors = {
    'nitrate': 0.95,    # Highly soluble, no precipitation
    'potassium': 0.90,  # Highly mobile, minimal binding
    'calcium': 0.85,    # Can precipitate at high pH
    'magnesium': 0.88,  # Generally stable in xylem
    'phosphate': 0.70,  # Can precipitate with Ca/Mg
    'sulfate': 0.92,    # Stable and soluble
    'iron': 0.60,       # Oxidation and precipitation issues
    'manganese': 0.75,  # Moderate stability
    'zinc': 0.80,       # Fairly stable
    'boron': 0.95       # Highly mobile as boric acid
}

base_efficiency = efficiency_factors.get(nutrient, 0.80)

# Concentration effects
if concentration > 20: # mM - high concentration may cause precipitation
    concentration_factor = 0.8
elif concentration < 1: # Very low concentration - minimal loss
    concentration_factor = 1.0
else:
    concentration_factor = 0.9

# pH effects (some nutrients precipitate at high pH)
ph = xylem_properties['ph']
if nutrient in ['phosphate', 'iron'] and ph > 6.5:
    ph_factor = 0.8
else:
    ph_factor = 1.0

return base_efficiency * concentration_factor * ph_factor
```

Phloem Transport: The Pressure Flow System

python

```

def model_phloem_transport_dynamics(source_strength, sink_demand, phloem_characteristics,
    nutrient_loads):
    """
    Model nutrient transport in phloem following pressure flow mechanism

    Phloem transport driven by:
    1. Osmotic pressure gradients (sugar concentration differences)
    2. Sink demand strength (metabolic activity, growth rate)
    3. Phloem loading/unloading efficiency
    4. Sieve tube conductivity and continuity

    The Münch pressure flow hypothesis:
    - High sugar concentration in source creates high osmotic pressure
    - Water influx increases hydrostatic pressure
    - Pressure gradient drives bulk flow toward lower-pressure sinks
    - Nutrients are carried along in this mass flow
    """

    # Phloem sap characteristics
    phloem_sap = {
        'sugar_concentration': 0.8, # M (very high - 15-25% by weight)
        'ph': 8.0, # Alkaline environment
        'osmotic_pressure': 2.5, # MPa (very high)
        'flow_velocity': 0.5, # m/hour (slower than xylem)
        'total_amino_acid_concentration': 50, # mM
        'typical_nutrient_concentrations': { # mM
            'potassium': 100,
            'magnesium': 10,
            'phosphate': 15,
            'sulfate': 8,
            'calcium': 2, # Very low - not readily mobile
            'nitrate': 5 # Usually as amino acids instead
        }
    }

    # Calculate pressure gradient
    pressure_gradient = calculate_pressure_gradient(source_strength, sink_demand)

    # Phloem flow rate
    phloem_flow_rate = calculate_phloem_flow_rate(
        pressure_gradient, phloem_characteristics['conductivity']
    )

```

```
# Nutrient transport in phloem
```

```
phloem_transport = {}
```

```
for nutrient, load_rate in nutrient_loads.items():
```

```
    # Loading efficiency into phloem
```

```
    loading_efficiency = calculate_phloem_loading_efficiency(nutrient, phloem_sap)
```

```
    # Concentration in phloem sap
```

```
    if phloem_flow_rate > 0:
```

```
        phloem_concentration = (load_rate * loading_efficiency) / phloem_flow_rate
```

```
    else:
```

```
        phloem_concentration = 0
```

```
    # Transport rate
```

```
    transport_rate = phloem_concentration * phloem_flow_rate
```

```
    # Unloading efficiency at sink
```

```
    unloading_efficiency = calculate_phloem_unloading_efficiency(  
        nutrient, sink_demand, phloem_sap  
    )
```

```
    # Net delivery to sink
```

```
    net_delivery = transport_rate * unloading_efficiency
```

```
    phloem_transport[nutrient] = {
```

```
        'loading_efficiency': loading_efficiency,
```

```
        'phloem_concentration': phloem_concentration,
```

```
        'transport_rate': transport_rate,
```

```
        'unloading_efficiency': unloading_efficiency,
```

```
        'net_delivery_to_sink': net_delivery,
```

```
        'transport_form': determine_phloem_transport_form(nutrient)
```

```
    }
```

```
return {
```

```
    'phloem_characteristics': phloem_sap,
```

```
    'pressure_gradient': pressure_gradient,
```

```
    'flow_rate': phloem_flow_rate,
```

```
    'nutrient_transport': phloem_transport,
```

```
    'transport_efficiency': calculate_overall_phloem_efficiency(phloem_transport),
```

```
    'limiting_factors': identify_phloem_limitations(phloem_characteristics, sink_demand)
```

```
}
```

```
def calculate_pressure_gradient(source_strength, sink_demand):
```

```
    """
```

Calculate pressure gradient driving phloem flow

Pressure gradient = f(Source osmotic pressure, Sink demand strength)

```
"""
```

Source pressure (proportional to photosynthetic rate and sugar accumulation)

```
max_source_pressure = 2.5 # MPa
```

```
source_pressure = max_source_pressure * min(1.0, source_strength)
```

Sink pressure (inversely related to sink demand)

```
min_sink_pressure = 0.5 # MPa
```

```
max_sink_pressure = 2.0 # MPa
```

```
sink_pressure = max_sink_pressure - (max_sink_pressure - min_sink_pressure) * min(1.0, sink_demand)
```

```
pressure_gradient = source_pressure - sink_pressure
```

```
return max(0, pressure_gradient) # Pressure gradient must be positive for flow
```

```
def calculate_phloem_loading_efficiency(nutrient, phloem_sap):
```

```
"""
```

Calculate efficiency of loading nutrients into phloem

Loading mechanisms:

1. Active transport (ATP-dependent)
2. Facilitated diffusion
3. Symplastic loading (through plasmodesmata)
4. Apoplastic loading (through cell walls and membranes)

```
"""
```

Nutrient-specific loading efficiencies

```
loading_efficiencies = {
```

```
    'potassium': 0.90,    # Efficient K+ channels and transporters
```

```
    'nitrate': 0.30,     # Poor - usually converted to amino acids first
```

```
    'phosphate': 0.80,   # Specific phosphate transporters
```

```
    'sulfate': 0.70,     # Moderate efficiency
```

```
    'magnesium': 0.75,   # Moderate - some binding issues
```

```
    'calcium': 0.15,     # Very poor - binding and precipitation
```

```
    'iron': 0.40,        # Chelation required for transport
```

```
    'zinc': 0.60,        # Moderate - chelate-dependent
```

```
    'manganese': 0.55,   # Moderate efficiency
```

```
    'boron': 0.85        # Good - as boric acid
```

```
}
```

```
base_efficiency = loading_efficiencies.get(nutrient, 0.50)
```

```

# pH effects on loading
ph = phloem_sap['ph']
if nutrient in ['phosphate', 'iron'] and ph > 7.5:
    ph_factor = 1.1 # Alkaline pH can help some nutrients
elif nutrient == 'calcium' and ph > 7.5:
    ph_factor = 0.8 # High pH causes Ca precipitation
else:
    ph_factor = 1.0

return base_efficiency * ph_factor

def determine_phloem_transport_form(nutrient):
    """
    Determine the chemical form in which nutrients are transported in phloem
    """

    transport_forms = {
        'nitrogen': ['glutamine', 'asparagine', 'arginine'], # Amino acids, not nitrate
        'phosphorus': ['phosphate', 'sugar_phosphates', 'ATP'],
        'potassium': ['K_ion'], # Free ion
        'magnesium': ['Mg_ion', 'Mg_ATP_complex'],
        'sulfur': ['methionine', 'cysteine', 'glutathione'], # Organic forms
        'calcium': ['Ca_chelates'], # Only when chelated
        'iron': ['Fe_chelates', 'nicotianamine_complexes'],
        'zinc': ['Zn_chelates', 'Zn_amino_acid_complexes'],
        'manganese': ['Mn_chelates'],
        'boron': ['boric_acid', 'sugar_borate_complexes']
    }

    return transport_forms.get(nutrient, ['unknown_form'])

```

Integration with Plant Physiology

Source-Sink Relationships and Nutrient Allocation

python

```
def model_source_sink_nutrient_dynamics(plant_organisms, growth_stage, environmental_conditions):
```

```
    """
```

Model how nutrients flow between plant organs based on source-sink relationships

Source-Sink Dynamics Change Throughout Development:

SEEDLING STAGE:

- Sources: Seed reserves, young photosynthetic cotyledons
- Sinks: Root development, shoot apical meristem

VEGETATIVE STAGE:

- Sources: Mature leaves (photosynthesis), roots (stored reserves)
- Sinks: Young expanding leaves, growing roots, stem elongation

REPRODUCTIVE STAGE:

- Sources: All photosynthetic tissues, remobilization from vegetative organs
- Sinks: Flowers, fruits, seeds (very strong sinks)

SENESCENCE STAGE:

- Sources: Senescing tissues (remobilization)
- Sinks: Storage organs, remaining active tissues

```
    """
```

Define organ characteristics

```
organ_characteristics = {}
```

```
for organ_id, organ_data in plant_organisms.items():
```

```
    organ_characteristics[organ_id] = classify_organ_source_sink_status(
        organ_data, growth_stage, environmental_conditions
    )
```

Calculate source capacities

```
source_capacities = {}
```

```
for organ_id, characteristics in organ_characteristics.items():
```

```
    if characteristics['status'] == 'source' or characteristics['status'] == 'source_sink':
        source_capacity = calculate_organ_source_capacity(
            organ_id, organ_data, characteristics, environmental_conditions
        )
        source_capacities[organ_id] = source_capacity
```

Calculate sink demands

```
sink_demands = {}
```

```
for organ_id, characteristics in organ_characteristics.items():
```

```
    if characteristics['status'] == 'sink' or characteristics['status'] == 'source_sink':
```



```
sink_demand = calculate_organ_sink_demand(
    organ_id, organ_data, characteristics, growth_stage
)
sink_demands[organ_id] = sink_demand
```

Nutrient allocation algorithm

```
nutrient_allocation = allocate_nutrients_by_priority(
    source_capacities, sink_demands, organ_characteristics
)
```

```
return {
    'organ_classifications': organ_characteristics,
    'source_capacities': source_capacities,
    'sink_demands': sink_demands,
    'nutrient_allocation': nutrient_allocation,
    'transport_pathways': map_nutrient_transport_pathways(organ_characteristics),
    'bottlenecks': identify_transport_bottlenecks(source_capacities, sink_demands)
}
```

```
def classify_organ_source_sink_status(organ_data, growth_stage, environmental_conditions):
```

```
    """
```

Classify each organ as source, sink, or transitional based on physiological status

```
    """
```

```
organ_type = organ_data['type']
organ_age = organ_data['age_days']
photosynthetic_rate = organ_data.get('photosynthesis_rate', 0)
growth_rate = organ_data.get('growth_rate', 0)
nutrient_content = organ_data.get('nutrient_content', {})
```

Age-based classification

```
if organ_type == 'leaf':
    if organ_age < 7:
        base_status = 'sink'    # Young expanding leaves
    elif organ_age < 30:
        base_status = 'source_sink' # Mature productive leaves
    elif organ_age < 50:
        base_status = 'source'    # Mature leaves
    else:
        base_status = 'senescing_source' # Old leaves remobilizing
```

```
elif organ_type == 'root':
    if growth_stage in ['seedling', 'early_vegetative']:
        base_status = 'sink'    # Active root growth
```

```

else:
    base_status = 'source_sink' # Storage and uptake functions

elif organ_type == 'stem':
    if growth_rate > 0.01: # Actively growing
        base_status = 'sink'
    else:
        base_status = 'source_sink' # Transport and some storage

elif organ_type == 'reproductive':
    base_status = 'strong_sink' # Flowers, fruits always strong sinks

else:
    base_status = 'source_sink' # Default

# Environmental modifications
if environmental_conditions.get('light_level', 400) < 200:
    # Low light reduces source capacity
    if 'source' in base_status:
        base_status = 'weak_source'

if environmental_conditions.get('temperature', 22) > 30:
    # High temperature increases maintenance demands
    if base_status == 'source':
        base_status = 'source_sink'

# Calculate source/sink strength
if 'source' in base_status:
    source_strength = calculate_source_strength(photosynthetic_rate, nutrient_content)
else:
    source_strength = 0

if 'sink' in base_status:
    sink_strength = calculate_sink_strength(growth_rate, organ_type, growth_stage)
else:
    sink_strength = 0

return {
    'status': base_status,
    'source_strength': source_strength,
    'sink_strength': sink_strength,
    'transport_priority': determine_transport_priority(base_status, growth_stage),
    'nutrient_mobility': assess_organ_nutrient_mobility(organ_data)
}

```

```
def calculate_source_strength(photosynthetic_rate, nutrient_content):
    """
    Calculate the capacity of an organ to supply nutrients to other organs
    """

    # Photosynthetic contribution (carbon + energy for transport)
    photo_contribution = photosynthetic_rate * 0.1 # Scaling factor

    # Stored nutrient availability
    mobile_nutrients = ['nitrogen', 'phosphorus', 'potassium', 'magnesium', 'sulfur']
    stored_nutrients = sum(
        nutrient_content.get(nutrient, 0)
        for nutrient in mobile_nutrients
    )

    # Remobilization capacity (depends on tissue age and composition)
    remobilization_capacity = stored_nutrients * 0.3 # 30% typically remobilizable

    total_source_strength = photo_contribution + remobilization_capacity

    return total_source_strength
```

```
def calculate_sink_strength(growth_rate, organ_type, growth_stage):
    """
    Calculate the demand strength of an organ for nutrients
    """

    # Base demand from growth rate
    base_demand = growth_rate * 10 # Scaling factor

    # Organ-specific demand multipliers
    organ_multipliers = {
        'leaf': 1.0,      # Standard demand
        'root': 1.2,      # Higher demand for active transport
        'stem': 0.8,      # Lower demand for structural tissue
        'reproductive': 2.0, # Very high demand
        'storage': 0.6     # Lower immediate demand
    }

    multiplier = organ_multipliers.get(organ_type, 1.0)

    # Growth stage effects
    stage_effects = {
```

```

    'seedling': 1.5,    # High relative demand
    'vegetative': 1.0,  # Normal demand
    'reproductive': 1.3, # Increased demand for reproduction
    'senescence': 0.5   # Reduced demand
}

```

```

stage_multiplier = stage_effects.get(growth_stage, 1.0)

```

```

total_sink_strength = base_demand * multiplier * stage_multiplier

```

```

return total_sink_strength

```

```

def allocate_nutrients_by_priority(source_capacities, sink_demands, organ_characteristics):

```

```

    """

```

Allocate available nutrients to competing sinks based on physiological priorities

Priority Hierarchy (from highest to lowest):

1. Reproductive organs (survival of species)
2. Root meristems (nutrient uptake capacity)
3. Shoot meristems (growth potential)
4. Young expanding leaves (future photosynthetic capacity)
5. Mature leaves (current photosynthetic maintenance)
6. Storage organs (reserve accumulation)
7. Structural tissues (maintenance only)

```

    """

```

Priority weights for different organ types and conditions

```

priority_weights = {
    'reproductive': 10.0,  # Highest priority
    'root_meristem': 8.0,  # Critical for nutrient uptake
    'shoot_meristem': 7.5, # Critical for growth
    'young_leaf': 6.0,     # Future productivity
    'mature_leaf': 4.0,    # Current productivity
    'expanding_organ': 5.0, # Active growth
    'storage_organ': 3.0,   # Reserve function
    'structural_tissue': 2.0 # Maintenance only
}

```

Calculate total available nutrients from all sources

```

total_available = {}
mobile_nutrients = ['nitrogen', 'phosphorus', 'potassium', 'magnesium', 'sulfur']

```

```

for nutrient in mobile_nutrients:
    total_available[nutrient] = sum(

```

```
    source_capacity.get(nutrient, 0)
    for source_capacity in source_capacities.values()
)
```

Calculate weighted demands

```
weighted_demands = {}
for organ_id, demands in sink_demands.items():
    organ_chars = organ_characteristics[organ_id]
    priority_weight = priority_weights.get(organ_chars['status'], 3.0)

    weighted_demands[organ_id] = {
        nutrient: demand * priority_weight
        for nutrient, demand in demands.items()
    }
```

Allocation algorithm

```
allocation_results = {}

for nutrient in mobile_nutrients:
    available_amount = total_available.get(nutrient, 0)
```

Calculate total weighted demand

```
total_weighted_demand = sum(
    demands.get(nutrient, 0)
    for demands in weighted_demands.values()
)
```

Allocate proportionally if demand exceeds supply

```
if total_weighted_demand > 0:
    allocation_factor = min(1.0, available_amount / total_weighted_demand)
else:
    allocation_factor = 1.0
```

Distribute nutrients

```
nutrient_allocation = {}
for organ_id, demands in weighted_demands.items():
    allocated_amount = demands.get(nutrient, 0) * allocation_factor
    nutrient_allocation[organ_id] = allocated_amount
```

```
allocation_results[nutrient] = {
    'total_available': available_amount,
    'total_demand': total_weighted_demand,
    'allocation_factor': allocation_factor,
    'organ_allocations': nutrient_allocation,
```

```
        'supply_adequacy': 'adequate' if allocation_factor >= 0.8 else 'limiting'  
    }  
  
    return allocation_results
```

This nutrient mobility model provides a comprehensive understanding of how plants orchestrate their internal nutrient economy, enabling precise management of plant nutrition for optimal growth and resource efficiency.