In [11]:
```
import re
import networkx as nx
import matplotlib.pyplot as plt
import json
```

In [12]:
```
# Load your card data (replace with actual data loading)
with open('data/labeled_cards.json') as f:
    cards = json.load(f)
```

In [13]:
```
def create_lookup_lists(cards):
    names = list(set(card['name'] for card in cards))
    traits = sorted(set(trait for card in cards for trait in card['traits']), ke
    colors = list(set(color for card in cards for color in card['color']))
    types = list(set(card['type'] for card in cards))
    return names, traits, colors, types

def parse_traits(effect, traits):
    return [trait for trait in traits if trait in effect]

def parse_max_cost(effect):
    match = re.search(r'cost (\d+) or (?:lower|less)', effect, re.IGNORECASE)
    return int(match.group(1)) if match else None

def parse_power_threshold(effect):
    match = re.search(r'power (\d+) or (?:lower|less)', effect, re.IGNORECASE)
    return int(match.group(1)) if match else None

def parse_card_names(effect, names):
    return [name for name in names if name in effect]

def parse_exclusions(effect, names):
    exclusion_phrases = ["other than", "except"]
    excluded = []
    for phrase in exclusion_phrases:
        if phrase in effect.lower():
            start_idx = effect.lower().find(phrase)
            exclusion_part = effect[start_idx:]
            excluded += [name for name in names if name in exclusion_part]
    return excluded

def parse_inclusions(effect, names):
    return [name for name in names if name in effect]

def parse_card_properties(effect, colors, types):
    props = {'colors': [], 'types': [], 'attributes': []}
    props['colors'] = [color for color in colors if color in effect]
    props['types'] = [ctype for ctype in types if ctype in effect]
    return props

def parse_leader_requirements(effect, names, traits, colors):
    return {
        'traits': parse_traits(effect, traits),
        'names': parse_card_names(effect, names),
        'multicolor': 'multicolor' in effect.lower() or 'multiple colors' in eff
    }

def create_synergy_graph(cards):
    G = nx.DiGraph()
```

```
    leaders = [card for card in cards if card['type'] == 'Leader']
    names, traits, colors, types = create_lookup_lists(cards)

    for card in cards:
        G.add_node(card['id'], label=f"{card['name']} ({card['id']})")

        if 'Searcher' in card['labels']:
            effect = card['effect']
            traits = parse_traits(effect, traits)
            included_names = parse_inclusions(effect, names)
            excluded = parse_exclusions(effect, names)
            max_cost = parse_max_cost(effect)
            props = parse_card_properties(effect, colors, types)

            for target in cards:
                if target['name'] in excluded:
                    continue
                if included_names and target['name'] not in included_names:
                    continue
                if traits and not included_names:
                    if not any(t in target['traits'] for t in traits):
                        continue
                if props['colors'] and target['color'][0] not in props['colors']
                    continue
                if props['types'] and target['type'] not in props['types']:
                    continue
                if max_cost and target['cost'] > max_cost:
                    continue
                G.add_edge(card['id'], target['id'], label='searches')

        if 'Summon' in card['labels']:
            traits = parse_traits(card['effect'], traits)
            max_cost = parse_max_cost(card['effect'])
            names = parse_card_names(card['effect'], names)

            for target in cards:
                if names and target['name'] not in names:
                    continue
                if traits and not any(t in target['traits'] for t in traits):
                    continue
                if max_cost and target['cost'] > max_cost:
                    continue
                if target['id'] != card['id']:
                    G.add_edge(card['id'], target['id'], label='summons')

        if 'Leader Locked' in card['labels']:
            req = parse_leader_requirements(card['effect'], names, traits, color

            for leader in leaders:
                match = True
                if req['names'] and leader['name'] not in req['names']:
                    match = False
                if req['traits'] and not all(t in leader['traits'] for t in req[
                    match = False
                if req['multicolor'] and len(leader['color']) <= 1:
                    match = False
                if not any(c in leader['color'] for c in card['color']):
                    match = False
                if match:
                    G.add_edge(card['id'], leader['id'], label='requires leader'
```

```
        if 'Removal' in card['labels']:
            max_cost = parse_max_cost(card['effect'])
            power_threshold = parse_power_threshold(card['effect'])

            for target in cards:
                if max_cost and target['cost'] <= max_cost:
                    G.add_edge(card['id'], target['id'], label='removes by cost'
                if power_threshold and target.get('power', 0) <= power_threshold
                    G.add_edge(card['id'], target['id'], label='removes by power
                if max_cost and 'Cost Reducer' in target['labels']:
                    G.add_edge(target['id'], card['id'], label='synergizes with
                if power_threshold and 'Debuff Power' in target['labels']:
                    G.add_edge(target['id'], card['id'], label='synergizes with

        if 'Discard' in card['labels'] or 'Mill' in card['labels']:
            for target in cards:
                if 'Trash Interaction' in target['labels']:
                    G.add_edge(card['id'], target['id'], label='synergizes with

        if 'Rest Character' in card['labels']:
            for target in cards:
                if 'Stun' in target['labels']:
                    G.add_edge(card['id'], target['id'], label='synergizes with

        if 'reveal the top card of your deck' in card['effect'].lower():
            for target in cards:
                if 'Rearrange Deck' in target['labels']:
                    G.add_edge(target['id'], card['id'], label='synergizes with

        if 'reveal 1 card from the top of your life cards' in card['effect'].low
            for target in cards:
                if 'Rearrange Life' in target['labels']:
                    G.add_edge(target['id'], card['id'], label='synergizes with

        if 'Don Minus' in card['labels']:
            for target in cards:
                if 'Don Ramp' in target['labels']:
                    G.add_edge(target['id'], card['id'], label='synergizes with

        if 'Take Life' in card['labels']:
            for target in cards:
                if 'Gain Life' in target['labels']:
                    G.add_edge(target['id'], card['id'], label='synergizes with

    return G
```

In [14]:

```
def export_graph(G, filename="graphs/card_synergies.graphml"):
    for node in G.nodes():
        G.nodes[node]['label'] = G.nodes[node].get('label', '')
        G.nodes[node]['type'] = G.nodes[node].get('type', '')
    nx.write_graphml(G, filename)

def visualize_graph(G):
    plt.figure(figsize=(20, 15))
    pos = nx.spring_layout(G, k=0.5, seed=42)
    nx.draw(G, pos, with_labels=True, node_size=2000, font_size=8, arrowsize=20,
    edge_labels = nx.get_edge_attributes(G, 'label')
    nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels, font_size=6)
    plt.show()
```

```
def export_json_graph(G, filename="graphs/card_synergies.json"):
    data = nx.node_link_data(G)
    with open(filename, 'w') as f:
        json.dump(data, f, indent=2)
```

In [15]:
```
synergy_graph = create_synergy_graph(cards)
```

In [16]:
```
# visualize_graph(synergy_graph)
```

In [17]:
```
export_json_graph(synergy_graph)
```

In [18]:
```
export_graph(synergy_graph)
```