CSE 202: Project Algorithmic Solutions and Analysis

Winter 2025

Melina Dimitropoulou Kapsogeorgou, Andrew Pan, Adrian Rosing, Andrew Russell, Benjamin Xia

Introduction

We chose to study *Power Grid*, a 2004 strategic board game by Friedemann Friese. It is a competitive, turn-based game with elements of supply and demand, network flow, and other unique components.

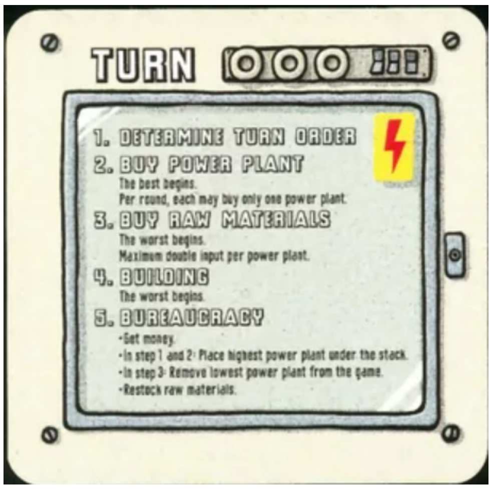
1. Global Constraints & Summary

The overall game objective for victory is to become the first player to power some $K \in \mathbb{N}$ cities, typically 15 cities in all. In our case, we will take this to mean minimizing the number of turns in order to attain K powered cities.

To do this, every turn we will be aiming to first maximize the number of cities that a player can power. If this objective is not possible (we cannot power K cities in the current round) we will be aiming to maximize the amount of money m_f that a player has available for future rounds.

In each turn, players will have some amount of money m to spend across all of the subphases, shown in the below image, which include buying power plants, buying resources, and buying cities. Players will earn B in power bills, (in our case) linearly proportional to the number of cities they have powered in any given round, with a universal basic income of $B_{ubi} \ll B$ if and only if a player doesn't power any cities for a round. The income scheme we will use will assume each powered city makes $B = \$10$, instead of the regular decaying amount of money earned in the game, discussed in Section 8 (Limitations).

During the course of the (unmodified) game there are 5 subphases, as seen in the below figure:


TURN

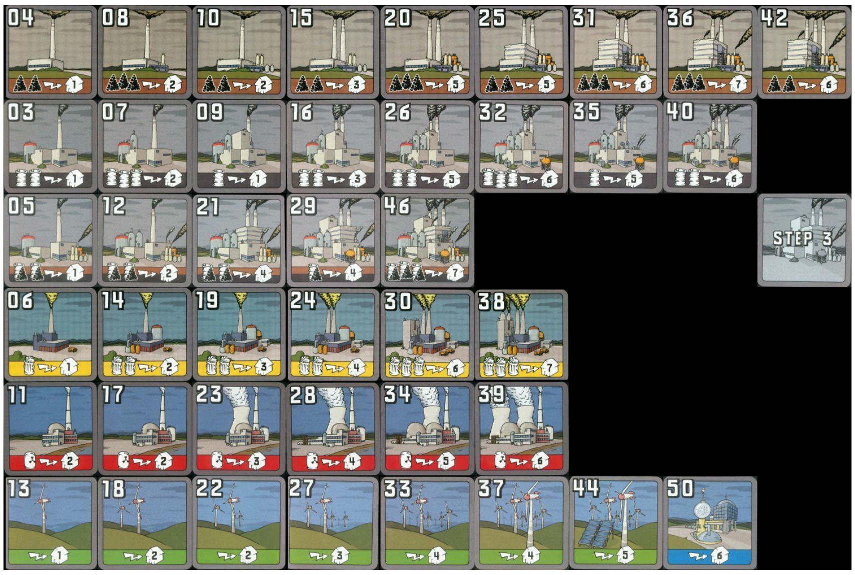
1. DETERMINE TURN ORDER
2. BUY POWER PLANT
   The best begins.
   Per round, each may buy only one power plant.
3. BUY RAW MATERIALS
   The worst begins.
   Maximum double input per power plant.
4. BUILDING
   The worst begins.
5. BUREAUCRACY
   •Get money.
   •In step 1 and 2: Place highest power plant under the stack.
   •In step 3: Remove lowest power plant from the game.
   •Restock raw materials.

Of these, we will be adjusting the game to make it more single player-like by:

1. Removing phase 1, the *Determine Turn Order* phase: We will be removing all multi-player elements and their difficult-to-model interactions from the game.
2. Keeping 2 (*Buy Power Plants*) as it is, while getting rid of the auction system.
2. Combining phase 3 (*Buy Materials*) and phase 4 (*Building*) for the sake of analysis: This decision will be discussed further below, but due to a lack of multiplayer elements in our limitations, we can do this without affecting our decisions.
4. Transforming phase 5 (*Bureaucracy*) into our global objective: Phase 5 is simply the cumulation of earlier phases, as it consists of powering cities and making money off of them. This is our global

goal, as discussed above.

As part of this, we will also be getting rid of the auction by replacing it with the minimum bet for each power plant (i.e. the number of the power plant). This number is seen on the top left of each plant below, and the number typically corresponds to how good each plant is and how efficient its resources are:



In addition to this, we will assume that all resources have infinite amounts of each resource (instead of a limited supply) and static prices (rather than prices determined by market demand). This will allow us to treat full turns (i.e. the full cycle of 5 phases/subrounds happening once) largely independently, since we will thus be assuming that we will not need to stockpile resources for future rounds. Thanks to this, in our analysis we will be able to combine phases 2 and 3, since buying power plants and resources will no longer be market fueled and thus can be done in conjunction to see what their joint cost is (i.e. what the total cost of buying a power plant would be).

Here is the price of each resource:

Resource Type	**Cost Per Resource**
Wind	$0
Coal	$2
Oil	$3
Garbage	$7
Uranium	$14

2. Game Constraints

Our game and algorithms assume the following additional constraints:

1. Powering cities is always more profitable than not powering cities (i.e. *Cost of powering a power plant* $<$ *Money earned if we were to power all houses within the power plant's capacity*). This is to avoid degenerate cases where the optimal decision would be to purely build roads and power **nothing** until a player has enough funds to power all K cities.

2. Each round can be treated fairly independently; i.e. doing a maximal turn in one round will not affect our ability to do a maximally optimal turn in a future round.

3. Each player's starting cash is sufficient to purchase a power plant in the first round (this is guaranteed if we remove the auction, but maintain the rest of the game as-is).

3. Pre-Game Initialization

Before the main game begins, each player must first choose some starting city, which we will call c_s. Since the game objective is to be the first player to power K (i.e. minimize rounds to power K cities), players will need to power additional cities to satisfy the overarching game objective. To connect more cities, each player must build connections from one of their currently connected cities C.

The game board is represented as a graph $G = (V, E)$, where each vertex is a city, and each undirected edge is a possible connection between cities. Each edge $e = (a, b) \in E$ has an associated cost $cost(e)$ with building a connection between cities a and b.

One method of determining a locally optimal starting city c_s, is to determine the cheapest cost to connect to $K - 1$ other cities (i.e. find a minimum spanning tree with K vertices such that the sum of edge weights $\sum_{j \in MST} cost(e_j)$ is minimized).

This is already a well-studied problem: The K-Minimum Spanning Tree Problem, which has already been proven to be an NP-Hard problem for variable K [1].

The best known polynomial time approximation algorithm: Garg's Algorithm [2] is able to approximate the theoretical optimal solution to within a factor of 2, in polynomial time for any arbitrary graph. Unfortunately, this (and other K-MST approximation algorithms) are extremely complex algorithms that go far beyond the scope of the class and what we believe is reasonable to explain. So we will instead brute force the optimal solution as follows.

Inputs:
- The original game graph $G = (V, E)$ where G is connected and each $e \in E$ has a non-negative cost associated with it.
- Some $K \in \mathbb{N}$, the number of vertices we want in our subgraph MST.

Output:
- Some $G' = (V', E')$ where $V' \subseteq V$, and $E' \subseteq E$ that forms the minimal K-Minimum Spanning Tree in G.

Algorithm/Runtime Analysis:

For all $\binom{|V|}{K}$ choices of vertices V', compute the MST that only makes use of these K vertices (i.e. discard all edges (a, b) where either $a \notin V'$ or $b \notin V'$, and $v \in V$ where $v \notin V'$ and compute the MST on the remaining graph). This takes $O((|V'| + |E'|) \log|V'|))$ runtime when using a Fibonnaci Heap, and since $|V'|, |E'| \in O(K)$, this gives us a runtime of $O(K \log K)$ for each MST computation.

Return the K-MST that has the minimal overall cost.

Thus our final runtime will be $O\left(\binom{|V|}{K} K \log K\right)$.

Correctness:

By correctness of Prim's Algorithm, each MST call is correct, and since we take the minimal of **all** possible K-MST's, our algorithm is trivially correct.

Now that we have our K-Minimum Spanning Tree, we, defining the optimal vertex to choose is a little harder to quantify, so we'll take the vertex $v \in V'$ with minimum sum of adjacent edge weights as our starting city.

The rest of our algorithm will only consider our K-MST G' for any kind of graph optimization problem.

Note: We are only considering optimality in the context of this pregame stage while ignoring later stages, which may favor fast expansion to other cities (and thus leading to more revenue from powering other cities) instead of minimal cost to connect K cities.

Comparison Against Baseline:

We present a baseline brute-force algorithm as follows:

- Generate all possible subgraphs of size K by iterating over all possible subsets of edges in the graph. This takes $O\left(2^{|E|}\right)$ time.
- Filter out any invalid subgraphs generated by checking if each edge subset forms a connected subgraph containing exactly K vertices. This can be done via depth-first search in $O(|V| + |E|)$ time.
- Compute the MST for each valid subgraph via Prim's algorithm, then return the lowest-weight graph found. Running Prim's takes $O(K \log K)$ time for each subgraph (as established above), and the lowest-weight graph can be stored while iterating over all $O\left(2^{|E|}\right)$ subgraphs.

This algorithm is correct because it iterates over every single subgraph of G and computes an MST. Therefore it must find the lowest-weight MST because the algorithm searches every possible (valid or invalid) MST that can exist in the graph. By way of search space exhaustion, this baseline algorithm must be correct.

The final runtime of this algorithm is $O\left(2^{|E|} \times ((|V| + |E|) + (K \log K))\right)$, which is significantly slower than the $O\left(\binom{|V|}{K} K \log K\right)$ runtime of our approach above. While both are exponential-time algorithms, the exponential term $O\left(\binom{|V|}{K}\right) = O(|V|^K)$ is significantly smaller than $O\left(2^{|E|}\right)$ assuming $K \ll |E|$. Therefore while our approach is already brute-force, we make more intelligent decisions to arrive at an optimal solution than the baseline brute-force approach.

4. *Purchase Power Plants* Phase

In this phase, players must decide which (if any) new power plants to acquire. As mentioned in our proposal, we simplify Power Grid's original auction behavior into a fixed-price sale, based on the card's number, for each plant. This reduces the complexity of the phase on its own, but maintains the overall cost-benefit considerations needed to succeed in later phases. We formulate the phase into a computational problem as follows:

Inputs:

- A set of power plants options P, where each plant $p = (c, r_t, r_a, e) \in P$ has a cost $c \in \mathbb{N}$, resource type $r_t \in \{$ Uranium, Coal, Oil, Trash, Wind $\}$ (this is just an example based on the game, our algorithm does not assume these resource types), resource amount $r_a \in \mathbb{N}$ (the amount of the resource r_t we need to power the plant for a round), and efficiency $e \in \mathbb{N}$ (number of cities the plant can power when activated).
- The player's available funds $f \in \mathbb{N}$.
- The player's current power plants P_{player}, where each plant $p = (c, r_t, r_a, e)$ as described above.
- Maximum plants a player may hold at a time $P_{max} \in N$ (set to 3 in the original game).
- A dictionary ResourceCost such that $r_t \to \mathbb{N}$, that tracks the price of each resource.

Output:

- A purchasing decision where the player buys a plant $p \in P$, or a null value (indicating that the player should pass).
- A discard decision $p \in P_{player}$ if the player buys a plant when $|P_{player}| \geq P_{max}$.

Constraints:

- The player's choice must fit within their budget, meaning they can only purchase $p = (c, r_t, r_a, e)$ where $f \leq c$.
- The player can only buy a single $p \in P$ per turn.
- If the player already has P_{max} power plants or more and decides to purchase another, they must discard one of their current power plants $p \in P_{player}$.
- We assume that we start off with 0 resources. Since resources are infinite and their prices are constant, we can get away with **only** purchasing resources we need for the current round.

Definition of (Local) Optimality:

For power plant purchasing phase, we will aim to maximize the number of cities we can theoretically power in the current round (completely ignoring whether we have enough cities connected). If there is more than one possible solution that can theoretically power at least K cities, we will pick the option that costs the least among them (while factoring in any cost of power plant purchases).

Algorithm:

If we have fewer than $P_{\max}$ power plants, we will need to consider $|P| + 1$ possibilities: We can purchase any one of $p \in P$ power plants, or we can choose to not purchase *any* power plant.

If we have $|P_{player}| = P_{\max}$ power plants, we will need to consider $P_{\max}|P| + 1$ possibilities: We can purchase any power plant, swap any of our current power plants out for a new plant, or make no changes to our roster of power plants.

Whatever the case, we will enumerate all possibilities and pick the best one according to our definition of optimality for this phase.

When we don't purchase any power plants, let the theoretical number of cities (along with cost) we can power be denoted by $(m, c) = \text{MaxPow}(P_{player}, f)$. We'll define the implementation of MaxPow subroutine later, but it will optimally return the maximum number of cities m we can theoretically power with the given configuration. In the case $m \geq K$, it will return the minimum cost configuration that powers at least K cities.

Likewise, let us define the solution when we purchase a power plant (without evicting one of our current plants) i as $(m, c) = \text{MaxPow}(P_{player} \cup \{p_i\}, f - c_i)$, and the case where we evict $q \in P_{player}$ and purchase plant i as $(m, c) = \text{MaxPow}(P_{player} \cup \{p_i\} \setminus q, f - c_i)$. When determining the best configuration, we will add the purchasing cost $c \leftarrow c + c_i$ during comparison for cases where we purchased a power plant.

Now, to determine the best configuration. As defined earlier, this will be the configuration that can theoretically power the most cities. When there are multiple configurations that can power at least K cities, then the best one among them will be the one that does so with minimum cost. Additional tiebreaking behaviors can also be defined on top of this but they are not necessary in the context of maximizing our objective for the current round/phase. One example could be picking the one with minimal cost when excluding power plant purchasing cost.

Time Analysis:

We iterate over a maximum of $O(|P|P_{\max})$ configurations, each taking $O(|P|f)$ time to compute MaxPow (as we will see below), so our final runtime is $O(|P|^2 P_{\max} f)$. This is an improvement over the brute-force runtime of $O(|P| \times 2^{P_{\max}})$, discussed below.

Proof of Correctness:

When assuming the correctness of MaxPow for each configuration of power plants (which we will prove later), we find the maximum number of cities each configuration can power, and if there are multiple configurations that can power at least K cities, we pick the cheapest one among them in the definition of our algorithm, which trivially satisfies our objective. Thus, our algorithm is correct.

Comparison Against Baseline:

A baseline algorithm for this phase is a brute-force set enumeration, which proceeds as follows: For each power plant $p \in P$ and the null purchase decision $p = \emptyset$, enumerate the set of all powering decisions given P_{player}.

That is, let P'_{player} be the set $(P_{player} \cup \{p\} \setminus p_l)$ if $|P_{player}| = P_{\max}$ (where p_l is the least efficient plant in P_{player}) or let $P'_{player} = P_{player} \cup \{p\}$ otherwise. Consider each combination of powering or not powering every $p' \in P'_{player}$, and discard any combination which exceeds f. Among the remaining combinations, select the one which maximizes the number of cities powered. If there is a tie, select the one which minimizes total cost.

After repeating this process for every $p \in P$ and $p = \emptyset$, return the purchase and discard decisions corresponding to the maximum number of cities powered over all combinations evaluated (breaking ties by minimum cost).

We know that enumerating a binary decision for all elements of a set takes $O\left(2^{|P_{player}|}\right)$ time, and this enumeration takes place $O(|P|)$ times. Because the final result of the algorithm can be stored and updated while enumerating sets, there is no additional overhead for determining the return value. If we assume in the worst case that $|P_{player}| = P_{\max}$, then this baseline algorithm has a runtime of $O(|P| \times 2^{P_{\max}})$.

Our algorithm has a runtime of $O(|P|^2 P_{\max} f)$, therefore we improve an exponential-time brute-force algorithm with a polynomial-time dynamic programming approach.

MaxPow Subroutine: Max Theoretical Cities Powered

For each configuration of power plants, we will compute the maximum number of power plants we can theoretically power as follows:

Inputs:
- A set of power plants P, where each plant $p = (c, r_t, r_a, e) \in P$ has a cost $c \in \mathbb{N}$, resource type $r_t \in \{$ Uranium, Coal, Oil, Trash, Wind $\}$, resource amount $r_a \in \mathbb{N}$, and efficiency $e \in \mathbb{N}$. Let P_i denote the i'th power plant in this set.
- The player's available funds $f \in \mathbb{N}$.
- A dictionary ResourceCost such that $r_t \to \mathbb{N}$, that tracks the price of each resource.

Outputs:
- The maximum number of cities we can theoretically power with the given power plants $MP \in \mathbb{N}$.
- The cash cost of powering said theoretical number of cities $MC \in \mathbb{N}$

For any power plant $p = (c, r_t, r_a, e)$, let $\text{RunningCost}(p) = r_a \times \text{ResourceCost}(r_t)$

For each power plant $p = (c, r_t, r_a, e)$, we have 2 possibilities:

1. We power the plant, and pay the full $\text{RunningCost}(p)$ (assuming of course, we can afford it.)
2. We leave it deactivated, and pay nothing (excluding any possible power plant purchase cost).

Algorithm:

This leaves us with solving two recursive subproblems: Returning the "best" (larger city count, if both city counts > k, then lesser total cost) of $\text{MaxPow}(P \setminus p_1, f)$ (not powering p_1) and $e_1 + \text{MaxPow}(P \setminus p_1, f - \text{RunningCost}(p_1))$ (powering p_1), where we only consider powering p_1 when $f - \text{RunningCost}(p_1) \geq 0$.

We use the base cases $\text{MaxPow}(\emptyset, *) = (0, 0)$, and $\text{MaxPow}(*, x) = (0, 0)$ for any $x \leq 0$ (0 cities powered, and 0 cost to power the plants).

This can be reformulated into a bottom-up dynamic program where $dp_{i,j}$ denotes the maximum number of cities we can (theoretically) power with the first i power plants and j cash in our budget. We also store the subsets of power plants $P'_{i,j}$ used to compute each $dp_{i,j}$, filled in alongside dp using set union instead of arithmetic. We do not use this array in this phase, but require it for following phases.

We can first solve our base subproblems $dp_{i,0} = dp_{0,j} = 0$ for any $i \in \{0, ..., |P|\}$ and $j \in \{0, 2, ..., f\}$.

Then we can solve recursive subproblems from rows $i = 1... \ |P|$, followed by columns $j = 1, ..., f$.

$$dp_{i,j} := \begin{cases} dp_{i-1,j} \text{ if } j - \text{RunningCost}(p_i) \leq 0 \\ \max\left(dp_{i-1,j}, dp_{i-1,j-\text{ RunningCost}(p_i)} + e_j\right) \text{ otherwise} \end{cases}$$

Subproblems will be solved from $dp_{1,1}$, then $dp_{1,2}, ..., dp_{1,f}$ followed by $dp_{2,1}, ..., dp_{2,f}, ..., dp_{|P|,f}$.

Finally, our solution will be $\left(\max_j\left\{dp_{|P|,j}\right\}, \text{argmax}_j\left\{dp_{|P|,j}\right\}\right)$ if $\max_j\left\{dp_{|P|,j}\right\} < K$. Otherwise, it will be $\left(\min_j\left\{dp_{|P|,j} \mid dp_{|P|,j} \geq K\right\}, \text{argmin}_j\left\{dp_{|P|,j} \mid dp_{|P|,j}\right\}\right)$.

Time Analysis:

The runtime of MaxPow will be $O(|P|f)$, as each subproblem computation takes $O(1)$ time and there are clearly $O(|P|f)$ subproblems (because the dp array has one axis determined by power plant number and another determined by total cash).

Proof of Correctness:

Claim: $dp_{i,j}$ correctly maintains the denotes the maximum number of cities we can (theoretically) power with the first i power plants and j cash in our budget.

Basis: When $i = 0$, or $j = 0$, the dynamic program is trivially correct as we have no cities to power or 0 cash to do so.

Induction:

Induction Hypothesis: All $dp_{a,b}$ where either $a < i$ or $b < j$ are correct.

Case: $\text{RunningCost}(p_i) \leq 0$, we cannot afford the power plant, so the maximum cities we can power will be the maximum of the remaining cities $dp_{i-1,j}$, which is correct by our induction hypothesis.

Case: $dp_{i-1,j} > dp_{i-1,j-\text{ RunningCost}(p_i)} + e_j$, then we can power more cities by **not** fueling the current plant, and setting $dp_{i,j} = dp_{i-1,j}$ is correct.

Case: $dp_{i-1,j-\text{ RunningCost}(p_i)} + e_j > dp_{i-1,j}$, then fueling the current plant lets us power more cities than not, thus this case is correct.

Thus $dp_{i,j}$ is correct for all i, j.

Now picking the cheapest option that powers at least K cities, or the option that powers most cities when that isn't possible, is trivially correct by our definition of MaxPow. Thus, MaxPow is correct.

5. *Purchasing Resources/Building Connections/Powering Plants* Phase

Since the next few phases could be combined, as they do not rely on other players' decisions or any external sources of randomness such as a slate of power plants available for purchase, we will combine the phases into one.

In the *Purchasing Resources* phase, players purchase resources. As stated earlier, all resources are available for purchase at a fixed price.

In the *Build Connections* phase, players select cities to build houses in. In the following phase, the player selects among these cities to supply with power. If the player builds into a city which is connected to an already-powered city, they must pay the connection cost listed on the edge between them. The number of connections that a player possesses also determines their turn order, potentially incentivizing spreading out across the map. We simplify this phase by assuming you can always build into "unowned" cities (i.e. the player only needs to pay the cost of building an edge to a city).

In the *Powering Plants* phase, players use their amassed resources (from the *Purchasing Resources* phase) to fuel plants, and power cities.

Inputs:

- The player's current power plants P_{player}, where each plant $p = (c, r_t, r_a, e)$ as described above.
- A graph $G' = (V', E')$ representing the Power Grid map (computed from our initial K-MST), where:
  - V' is the set of cities
  - E' is the set of undirected edges between cities, each with a weight $w : E \to \mathbb{R}^+$ representing connection costs
- The player's current set of connected cities $C \subseteq V'$
- The player's available funds $f \in \mathbb{N}$.
- A dictionary ResourceCost such that $r_t \to \mathbb{N}$, that tracks the price of each resource.

Output:

- A set of resource purchase decisions (r_t, q) where r_t denotes the resource type and $q \in \mathbb{N}$ is the quantity.
- A set $e \subseteq E$ edges to connect to their current network.
- A set $P' \subseteq P_{player}$ of power plants to power.

Constraints:

- The player must not exceed their available funds f when purchasing resources, building connections, and activating power plants collectively.

$$\sum_{e_i \in E_{\text{purchased}}} w(e_i) + \sum_{(r_t, q) \in R_{\text{purchased}}} \text{ResourceCost}(r_t) \cdot q \leq f$$

- The player cannot power more cities than are connected in the network.

$$|C| \leq |P_{\text{player}}|$$

Definition of Optimality:

As in the *Purchase Power Plants* phase, we will optimize for the global objective. That is, maximize cities powered in the current round. If that number is less than K, maximize the amount of money that the player finishes the round with.

Algorithm:

First we must determine the maximum number of cities we can power in the current round. In order to accomplish this do the following:

1. Run Prim's Algorithm from our current vertices C to get an ordering of edges we can add from our current cities. Let this ordering be denoted as PO.
2. Get a prefix sum of the edge costs in Prim's Algorithm's ordering. $\rightarrow$ Prefix_i represents the cumulative cost of adding the first i edges, compute for all i starting from index 0, to index n inclusive for $n + 1$ total indices.
   Note: to get a truly optimal ordering, we would need to brute force this to a certain extent (discussed later).
3. Run MaxPow(P, f) program to determine the maximum number of cities we can power and cost (MP, MC). Maintain the $dp_{i,j}$ matrix as defined in the MaxPow subroutine, defined as $dp_{i,0} = dp_{0,j} = 0$ for any $i \in \{0, ..., |P|\}$ and $j \in \{0, 2, ..., f\}$.

$$dp_{i,j} := \begin{cases} dp_{i-1,j} \text{ if } j - \text{RunningCost}(p_i) \leq 0 \\ \max\left(dp_{i-1,j}, dp_{i-1,j- \text{ RunningCost}(p_i)} + e_j\right) \text{ otherwise} \end{cases}$$

4. For each Prefix_i from $i = 1, ..., |V' \setminus C|$ inclusive, get $dp_{|P|,f-\text{Prefix}_i}$. The maximum number of cities we can power will be:

$$M = |C| + \text{argmax}_i \min\left(dp_{|P|,f-\text{Prefix}_i}, |C| + i\right)$$

5. If $M \geq K$, do the following:

- Take the subset of PO corresponding to the optimal i found in step 4. That is:

$$i^* = \min\left(dp_{|P|,f-\text{Prefix}_i}, |C| + i\right)$$

$$e = \{PO_k \mid 0 \leq k \leq i^*\}$$

- Extract the power plants contributing to $dp_{|P|,f-\text{Prefix}_i}$ via MaxPow's $P'_{|P|,f-\text{Prefix}_i}$ array. Let the newly-acquired plants be $P_{new} = P'_{|P|,f-\text{Prefix}_i} \setminus P_{player}$.
- Purchase enough resources to power each plant in P_{new} via:

$$R_t = \{r_t \mid p = (c, r_t, r_a, e) \in P_{new}\}$$

$$Q = \{r_a \mid p = (c, r_t, r_a, e) \in P_{new}\}$$

- Return $(R_t, Q), e$, and P_{new} as output.

6. Otherwise if $M < K$, optimize for the cash available in the next round. This is given by:

$$\max_i \max_{0 \leq j \leq f - \text{Prefix}_i} f - \text{Prefix}_i - j + \min\left(dp_{|P|,j}, |C| + i\right)B$$

j is cash spent on resources, B is cash per city powered, i is number of new cities connected.

- Compute the edge set from PO similarly:

$$i^* = \arg\max_i \max_{0 \leq j \leq f - \text{Prefix}_i} f - \text{Prefix}_i - j + \min\left(dp_{|P|,j}, |C| + i\right)B$$

$$e = \{PO_k \mid 0 \leq k \leq i^*\}$$

- Retrieve the power plants $P'_{|P|,j}$ generated during MaxPow. Let the set of newly-acquired plants be $P_{new} = P'_{|P|,j} \setminus P_{player}$.
- Purchase enough resources to power each plant in P_{new} via:

$$R_t = \{r_t \mid p = (c, r_t, r_a, e) \in P_{new}\}$$

$$Q = \{r_a \mid p = (c, r_t, r_a, e) \in P_{new}\}$$

- Return (R_t, Q), e, and P_{new} as output.

Proof of Correctness:

It's important to note that we are using a heuristic for the ordering of edges here, as this is known as the rooted K-MST problem, and as stated earlier, solving for it optimally either takes exponential time, or can be approximated by very complex primal-dual algorithms (that do fancy stuff like assigning energy to vertices, and then iterating to minimize the difference in energy between vertices) that are far beyond the scope of the course.

We will show the remaining portion of our algorithm is optimal when the edge construction ordering is fixed as follows.

In order to prove that the algorithm correctly maximizes the number of cities powered in the current round, or if that number is less that K, optimize the amount of cash available for the next round, we rely on the following lemmas:

- Correctness of MaxPow subroutine's $dp_{i,j}$. This is proven in the previous section. We have already shown that the $dp_{i,j}$ construction accurately reflects the maximum number of cities that can be powered from the i first cities in the ordering and j budget.
- Correctness of combining results from PO and MaxPow to maximize the number of cities powered.

Note that Prefix_i denotes the cumulative cost of adding the first i edges to MinConnectC's ordering. Therefore, at each i, the remaining budgets (for buying resources) is $\text{Remaining}_i = f - \text{Prefix}_i$. By the correctness of MaxPow, $dp_{|P|,f-\text{Prefix}_i}$ must reflect the maximum number of cities that can be powered using the remaining budget at i in the dynamic programming subroutine.

Consider an arbitrary i number of cities connected . $dp_{|P|,f-\text{Prefix}_i}$ must reflect the maximum number of cities that can be powered at this i. $|C| + i$ reflects the total number of cities that can be connected at this iteration. Therefore, the equation $\min\left(dp_{|P|,f-\text{Prefix}_i}, |C| + i\right)$ reflects the smallest of these two values, as a player cannot possibly power more cities than are connected, or power plants without money to buy the resources (satisfying the two constraints of the problem). Therefore $\arg\max \min\left(dp_{|P|,f-\text{Prefix}_i}, |C| + i\right)$ will select the value of i (number of new connections) that maximizes the total number of cities powered, and $|C| + \arg\max \min\left(dp_{|P|,f-\text{Prefix}_i}, |C| + i\right)$ reflects the total number of cities powered after the turn.

- Correctness of the cash optimization step.

The cash available in this next step can be calculated as the total budget minus spendings (to connect cities and buy resources) plus money earned (from connected cities). This can mathematically be expressed as:

$$\text{Cash} = f - \text{Prefix}_i - j + \min\left(dp_{|P|,j}, |C| + i\right)B$$

where j is cash spent on resources, B is cash reward per city powered, and i is the number of new cities connected. This is exactly what our algorithms cash optimization step computes. The algorithm uses the max operator over j to maximize the cash saved, and then maximizes over i to select the configuration with most connections out of all “ties” for most money in the next round. The max operator over j ensures that, among all possible allocations of resources, we retain the highest possible leftover cash while still achieving the optimal number of powered cities.

Therefore, the algorithm is correct because 1) Prim’s makes the locally optimal ordering of cities for maximum connections, 2) MaxPow correctly computers the maximum cities that can be powered given the remaining budget, 3) constraints are correctly balanced, and 4) the cash optimization step ensures that in the case that cash is the optimization target, cash amount on next turn is correctly calculated. Therefore we have shown the algorithm optimizes targets as expected.

Time Analysis:

We begin by running Prim’s algorithm, which we know has a runtime of $O(|E'| \log|V'|)$ when using a binary heap and $O(|E'| + |V'| \log|V'|)$ when using a Fibonacci heap. We then compute a prefix sum over the edges in PO, which takes $O(|PO|)$ time. We have established that the MaxPow subroutine has a pseudo-polynomial runtime of $O(|PO|\ f)$, and computing M from the dp array takes $O(|PO|)$ time. In the $M \geq K$ case, computing P_{new} takes $O(|P|)$ time (assuming non-constant time set difference), computing e takes $O(|PO|)$ time, computing R_t and Q both take $O(|P|)$ time.

If we assume a binary heap for Prim’s algorithm, the overall runtime of our algorithm is $O(|E'| \log|V'|) + O(|PO|) + O(|PO|\ f) + O(|P|)$. We note that $|PO| = O(|E'|)$ in the worst case, but we cannot state that the $O(|E'|\ f)$ term dominates $O(|E'|)$ because it may be true that $f = 0$. Thus, we simplify our final runtime to become $O((|E'| \log|V'|) + |E'|\ f + |E'|)$.

If we assume a Fibonacci heap for Prim’s algorithm, the overall runtime of our algorithm is $O(|E'| + |V'| \log|V'|) + O(|PO|) + O(|PO|\ f) + O(|P|)$. Via the above simplifications, this runtime becomes $O(|V'| \log|V'| + |E'|\ f + |E'|)$.

Comparison Against Baseline:

We compare our algorithm to a brute-force baseline which exhaustively enumerates all possibilities:

- Enumerate all subsets of edges S such that:
  - S forms a connected graph extending C, meaning all new cities connect back to C. Let $C(S)$ be the set of cities reachable after adding S.
  - $\sum_{e \in S} w(e) \leq f$, meaning the total cost of edges is affordable.
- For each possible city expansion S, enumerate all subsets of power plants $P' \in P_{player}$ that can be activated while staying within budget:
  - Compute the cost to run P', denoted $\text{RunningCost}(P')$.
  - Compute the number of cities P' can power: $\text{PoweredCities}(P') = \sum_{(c, r_t, r_a, e) \in P'} e$.
  - Ensure $\text{RunningCost}(P')$ is less than or equal to the remaining funds after building edges.
- Enumerate all valid resource allocations:
  - For each power plant $p = (c, r_t, r_a, e) \in P'$, enumerate all possible resource purchases which fit within $f - c$. Let any given allocation be denoted as R.
- For each combination of (S, P', R):
  - Compute the number of cities powered $M = \min(\text{PoweredCities}(P'), |C(S)|)$.
  - Compute the remaining cash after all decisions: $f_{next} = f - \sum_{e \in S} w(e) - \text{RunningCost}(P')$.
  - Score using the objective function:
    - If $M \geq K$, maximize M.

  - Otherwise, maximize f_{next}.
- While enumerating, keep track of the best solution found and return it once complete.

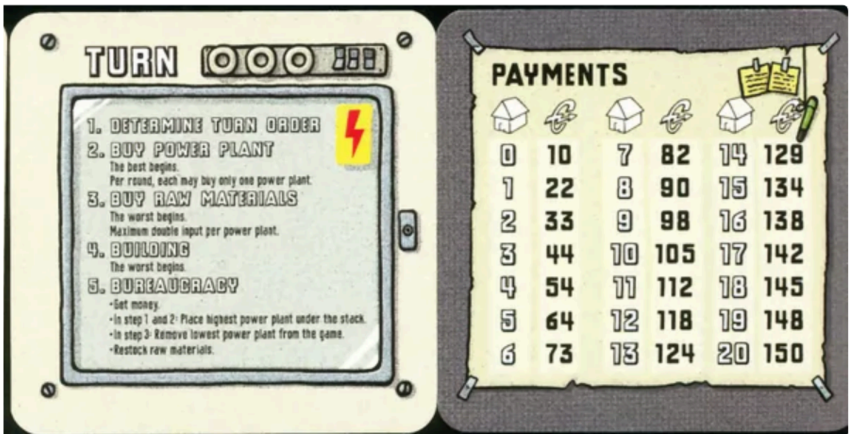
This brute-force algorithm enumerates $O\left(2^{|E|}\right)$ edges, $O\left(2^{|P_{player}|}\right)$ power plants, and $O\left(f^{|P'|}\right)$ resource allocations (with constant time operations in each enumeration). Because we enumerate over each set together, the overall time complexity is the product of all three. This implies that this brute-force approach has runtime $O\left(2^{|E|} \times 2^{|P_{player}|} \times f^{|P'|}\right)$.

This implies that our algorithm's binary heap-based runtime of $O((|E| \log|C|) + |E|\ f + |E|)$ is a significant improvement, from an exponential runtime to a pseudo-polynomial runtime where f tends to be small (on the order of 100) according to the official rule book. While a pseudo-polynomial runtime is effectively also an exponential runtime, we note that we no longer have three separate exponential terms (and that the brute-force approach also has a pseudo-polynomial term in $f^{|P'|}$).

6. Limitations

For completeness, we present a number of limitations and further simplifications made when designing our algorithmic solutions. We will discuss each and why they still allow our model to be a good representation of actually playing the game:

- We treat each turn independently: The player can't sacrifice a round to save up money to maximize the next round.
  - When playing the game, the rounds tend to be very independent; due to the auction system, it becomes challenging to predict what to save money for, since someone may snatch it up before you get to it. Thus, focusing on the goal of the game and making as much money as possible (by powering more cities) on a turn-by-turn basis is a fair focus, even if it may not necessarily be
- Simplified the way money is made from cities to be linear in the number of cities, original game's approach below:
  - The game's original distributions can be seen below. We made this simplification to simplify the analysis. However, once it is implemented, it would not be challenging to simply use a lookup table to adjust the algorithms instead of a linear lookup. The one additional complication arising from this linear assumption (and another assumption, that we should buy as many houses as possible) is the decay in money making. However, this decay is low enough that by the time we reach 9 houses, where the drop-off goes below a linear assumption, the game is normally done within a turn or two, so the $3-4 difference does not make a large enough change.

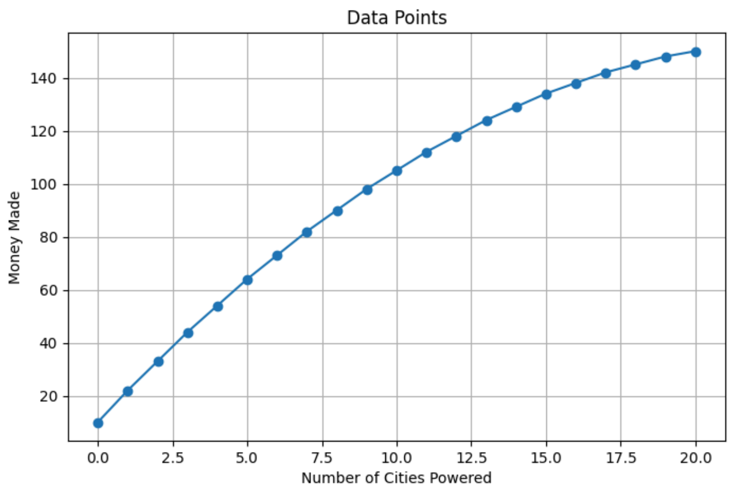


TURN

1. DETERMINE TURN ORDER
2. BUY POWER PLANT
   The best begins.
   Per round, each may buy only one power plant.
3. BUY RAW MATERIALS
   The worst begins.
   Maximum double input per power plant.
4. BUILDING
   The worst begins.
5. BUREAUCRACY
   •Get money.
   •In step 1 and 2: Place highest power plant under the stack.
   •In step 3: Remove lowest power plant from the game.
   •Restock raw materials.

PAYMENTS

0	10	7	82	14	129
1	22	8	90	15	134
2	33	9	98	16	138
3	44	10	105	17	142
4	54	11	112	18	145
5	64	12	118	19	148
6	73	13	124	20	150



- Remove multi-player actions, such as the auction system (replaced by the direct purchase system) and the negative effects of being ahead in the game.
  - While this is debatabley the largest change we made, it was necessary as the actions players take throughout the game are incredibly challenging to accommodate with a static algorithm. In order to accommodate such an environment, we would likely be best off using reinforcement learning. With that said, our algorithms would largely work if we introduce dynamic pricing for each auction and resource depending on the current state of the game when it gets to the player. As such, the algorithms we wrote would largely work in a multi-player game, with some minor tweaking.
- Change the way resource supply and demand works: Infinite resources, no changing prices. The original game gives a preset (non-linear) amount of cash for each city powered, we made a simplifying assumption that each city pays the same amount (though our solution should be easily adaptable to this case). In addition to this, we removed keeping & buying excess resources: Normally, the player can store up to two times the amount of the resource a power plant needs.
  - This is parallel to the above two points; by doing a mixture of our approach of simulating the amount of money made by a differing number of cities and making pricing dynamic, we can retain our current algorithm and it should work, although these are hard to model since they depend on having knowledge of what actions other players decide to take.
- Remove the cost of cities changing & other people blocking cities in by already settling in them.
  - This is just a constant number that we could coordinate into our second algorithm, by adjusting the cost of cities depending on who settled there. With this adjustment, our algorithm would work in a multi-player setting.
- Remove the separated phase system: Normally, there are 3 phases to the game (dependent on how far people have progressed).
  - As discussed in each of the subproblems, since we have removed the multi-player functionality of the game, the phase ordering no longer matters, as long as we buy plants and resources before powering (which our ordering of the subproblem does). Thus, this does not affect the results.

Bibliography

[1] R. Ravi, R. Sundaram, M. V. Marathe, S. S. Ravi, and D. J. Rosenkrantz, “Spanning trees short or small.” [Online]. Available: https://arxiv.org/abs/math/9409222

[2] N. Garg, “Saving an epsilon: a 2-approximation for the k-MST problem in graphs,” in *Proceedings of the Thirty-Seventh Annual ACM Symposium on Theory of Computing*, in STOC '05. Baltimore, MD, USA: Association for Computing Machinery, 2005, pp. 396–402. doi: 10.1145/1060590.1060650.