Hindawi Publishing Corporation
International Journal of Computer Games Technology
Volume 2015, Article ID 975371, 10 pages
http://dx.doi.org/10.1155/2015/975371

Research Article

Quality Measures for Improving Technology Trees

Teemu J. Heinimäki and Tapio Elomaa

Department of Mathematics, Tampere University of Technology, P.O. Box 553, 33101 Tampere, Finland

Correspondence should be addressed to Teemu J. Heinimäki; teemu.heinimaki@tut.fi

Received 23 October 2014; Accepted 22 March 2015

Academic Editor: Yiyu Cai

The quality of *technology trees* in digital games can be improved by adjusting their structural and quantitative properties. Therefore, there is a demand for recognizing and measuring such properties. Part of the process can be automated; there are properties measurable by computers, and analyses based on the results (and visualizations of them) may help to produce significantly better technology trees, even practically without extra workload for humans. In this paper, we introduce useful technology tree properties and novel measuring features implemented into our software tool for manipulating technology trees.

1. Introduction

Skill progression of player characters (PCs), acquiring new talents, achieving perks, and developing available technologies are essential parts in many popular digital games. Mental, physiological, and material upgrades significantly affect the narration of a game. They also partly establish the game logic and make it visible and understandable for the player. When players are able to grasp the intended logic, a feeling of more logical consequences follows. It leads to more profound user immersion and increases the enjoyment of play. The upgrades can also be used to partition the game horizon into separate stages or eras. Furthermore, the accumulating abilities set short-term goals for the player.

Technology trees (or *tech trees* for short), as they are called especially in strategy games, are structures used routinely to model and implement these aspects of gameplay. They describe and define the dependencies of technological (in a very broad sense) progress followed in the game. In other game genres, tech trees are called with different names such as *skill trees* or *talent trees*. Also within the genre of strategy games there is some variance in the terminology; for instance, the term *tech web* is sometimes used. In this work, however, we categorize all such structures under the common concept of tech trees.

Despite the central status of tech trees in modern (real-world, commercial) games, our experience and a survey show that they suffer from many deficiencies in practice. In this work, we demonstrate that weaknesses in tech trees can be automatically screened and detected by using suitably defined quality measures. That opens up the possibility to rectify the defects that might have been introduced to tech trees unintentionally. Our vision is that in the future automatic improvement possibilities offered by generic tools will be available for tech trees failing to meet the given quality criteria. The primary reasons for striving for such automation are our aim to improve efficiency and the desire to overcome the human limitations and guarantee a high game quality. One should also remember the fact that a game designer is not necessarily a programmer [1]. Different easy-to-use software tools are already used for many purposes in real-world game development. They have often a considerable positive impact on the quality of the end products, so why not create and use tools also to construct, measure, and adjust tech trees?

The main contribution of this work is the introduction of several measures for monitoring the quality of technology trees. In addition, a general-purpose software tool called Tech Tree Tool (TTT) [2] is improved by implementing quality monitoring. In order to empirically qualify the measures, we determine them, as applicable, for *Sid Meier's Civilization V* (Firaxis Games, 2010) (Civ 5) and discuss the results.

Even though we use Civ 5 as our default example, the results and observations are by no means restricted to this particular game or even the genre it represents. We could as

well have used, for instance, *Kerbal Space Program* (Squad, Beta Version 0.90.0 2014) as an illustrative example; it is a quite different game: a space flight simulator. However, in order to expose the illustrations to as wide an audience as possible, we stick to Civ 5 as our running example.

Next, in Section 2, we focus on the nature of technologies and characterize tech trees. We continue in Section 3 by pointing out some typical problems within them. After that, we introduce the measures that we find usable as indicators in quality assurance, first for technology trees (in Section 4) and then for individual technologies (in Section 5). Thereafter, Section 6 sheds some light on our automated measuring implementation, which is then tested with Civ 5 in Section 7. Finally, Section 8 discusses briefly the matter of adjusting and improving tech trees based on measurements, and Section 9 concludes the paper.

2. The Nature of Technologies and Technology Trees

Tech trees have become essential structures forming crucial mechanisms in digital games. There are different kinds of tech trees, and they "fulfill various strategic and narrative functions" [3]. Traditionally, technology trees have been associated with strategy games, of which most feature them [4], but effectively similar constructs are used also within other game genres. A classical example is *Diablo II* (Blizzard North, 2000) that is often mentioned as the game that truly introduced such structures within the genre of computer role-playing games. The idea of the "character skill tree" of the game was based on technology trees of strategy games [5]. Adoption to different genres and subgenres has led to variance in terminology; a tech tree-like structure may come under the disguise of a different name, but the difference is essentially only in the terms used for the items (e.g., "technologies," "perks," or "talents") that can be selected to be developed or purchased. The targets of their effects also vary; for instance, "talents" typically affect individual characters and "technologies" larger entities such as tribes, nations, or species. However, the basic idea of the structures remains the same. In this paper, we consider all these variants as technology trees. Correspondingly, those selectable items are henceforth called *technologies* or shortly *techs*.

A tech tree can be seen, for instance, as "a structure that controls progress from one technology to a better technology, enabling the player to create better facilities or more powerful units" ([6], page 141), or simply "a flow-chart showing the dependencies of upgrades and buildings" ([6], page 72). These views reflect the main two sides of the coin called technology tree; tech trees are typically seen either as (1) structures (mechanisms) for defining and controlling development (upgrading), based on dependency relations of technologies, or (2) flow-chart-like presentations on these dependencies. In this paper, we concentrate more on the former point of view.

Tech trees are not usually trees in graph-theoretical sense but can take forms of different, typically acyclic and weighted, (multi)digraphs. It is natural to treat technologies as nodes of a graph and define their relations using weighted edges. (In order to be able to handle different technology trees using a general characterization, we use a derivative of this basic approach: we basically consider weights to be associated with sets of edges. Sections 4 and 6 illuminate this more.) In addition to these basic components of any graph, additional data, for instance, information related to presentation or effects of the technologies, are typically needed.

Games come in various forms and flavours, and so do technologies used in them. For instance, personal properties, perks, traits, feats, and talents of game agents (e.g., PCs) can be seen and modeled as technologies, if the agents can learn or otherwise acquire them. Such techs are often found in games featuring roleplaying elements. For instance, a thievery-oriented PC might be improved by making it learn *Basic Lockpicking* or, already having acquired, say, *Improved Stealth*, advancing it to *Stealth Mastery*.

In strategy games, on the other hand, the players typically act as commanders, "controlling things from a discrete distance" [7]. This is reflected on the "traditional tech trees" of these games: the effects of technologies target typically larger wholes than individual characters.

Even when discussing these original tech trees, the word technology is used quite liberally: not all "technologies" found in strategy games are technical in nature at all. A player might, for example, develop *Banking* in order to bolster the economy of the controlled nation, invent *Free Time* to raise the general morale, or let the citizens learn *Ghuli'Xi'ulian Language* to be able to communicate with a neighboring game faction. These are *abstract technologies* [8]; their manifestations do not correspond to visible unit instances "on the board." Abstract techs can represent, for instance, cultural or administrative innovations and achievements. Various subtypes, like forms of government, religions, ideological movements, monuments, and pieces of art, fall into these categories. However, abstract techs can also contain technologies, technical devices, and technological inventions in the conventional sense (such as *Printing Press* or *Jet Engines*).

Sometimes, technologies represent different unit types (like samurai, healer, or dragon) used in the game, or, more accurately speaking, the abilities to produce units of the corresponding types. For instance, developing a technology called *Fluffy Bunny* could let the player produce fluffy bunnies (visible unit instances of the type "fluffy bunny") within the game world (given that other possible prerequisites, e.g., being able to pay the production costs and having necessary production facilities, are met). A single technology may also have several effects of possibly various natures; it is possible to have techs enabling unit production and giving simultaneously also other abilities [9].

Similarly, building types (if buildings are not treated as units) may have their specific construction-enabling technologies, and technologies can, in addition to their other effects, allow constructing buildings of certain types. Especially in real-time strategy games, the capability to construct buildings often depends on the types of the buildings already built. Buildings may enable "developing technologies"

in them, but also the dependencies between building types can be modeled by technology trees.

Technology trees serve many purposes. According to Sid Meier, undoubtedly the most famous computer game designer in the western world, the addictiveness of *Sid Meier's Civilization* (MicroProse, 1991) is at least partially due to "interesting decisions" [10]. A tech tree is certainly one of the most obvious ways to provide a player with possibilities to make such decisions. In addition to functioning as an upgrading system, tech trees can be used to represent technological history, as release mechanisms, and as narrative tools [3]. The decisions made by a player are often goal-oriented: the overall goal is typically victory, but acquiring technologies sets also short-term goals. Even if a technology does not have any significant impact on the gameplay, having it developed may, nevertheless, be something to brag about (see the article by Gazzard [11] concerning different rewards). As Huizinga ([12], page 50) put it, "in all games it is very important that the player should be able to boast of his success to others."

3. On the Demand for Monitoring Tech Tree Quality

Various properties affect the observed quality of a technology tree, and many common defects may reduce it. In order to find features typically considered problematic, we have been conducting an informal (unpublished) survey since 2012 by observing several WWW discussion forums. Because of the nature of the data, they may be biased; they have been acquired from a (somewhat arbitrary) set of forums, some of which concentrate on specific games (the sites we have been monitoring include, e.g., http://www.quartertothree.com/game-talk/, http://forums.2k.com/, http://forums.civfanatics.com/, http://www.gamespot.com/forums/, and http://forums.steampowered.com/forums/). Occasionally, also interpreting and categorizing the actual problems based on web publications may be problematic. Because of these difficulties, we do not announce any exact results, but only claim that by analyzing hundreds of writings (by various authors) we have found several problems that seem significant based on the frequency of them being mentioned or otherwise. We leave it as future work to validate the importance of the identified defect types.

Based on this survey, our experience, that of some of our fellow gamers, and WWW articles found using search engines, the most typical problems concerning existing tech tree implementations in real games seem to include

(i) too small or too large number of technologies,

(ii) too few requirements for developing technologies (allowing bypassing techs in an undesirable way),

(iii) too obvious *strong paths* (not encouraging to explore the tree, but only to exploit the known efficient strategy),

(iv) poor balancing (temporal or resource-wise),

(v) too limited possibilities to explore the tech tree during a game,

(vi) typically fixed and rigid (predefined) structure without any possibility of temporal variation or, for instance, variation between different game instances,

(vii) meaningless technologies (with only minor effects not motivating to advance in a tech tree),

(viii) requirements that do not make sense semantically, thematically, or historically.

The list presented here is not intended to be generally comprehensive. Grievances concerning visualization, aesthetics, and interaction issues are omitted (although common), because our focus lies on the structure and functionality, not on user interface (UI) details.

Of course, the quality of a tech tree is a highly subjective matter, and adjusting tech tree properties is all about making tradeoffs. However, in order to make justified adjustments to any direction, one must be able to evaluate tree properties. Common concerns serve as a good guideline for deciding which features are important. Balancing tech trees in terms of time and (other) resources is a generally meaningful problem. In this paper, we focus on such balancing, since somewhat objectively measurable tech tree features affecting the issue can be found. Balancing can be either *internal balancing* (within a tech tree) or *balancing between different tech trees*.

One typical central function of a technology tree is to control game flow with respect to time; tech trees are used for so-called gating and for controlling the narration or the overall "big picture game experience." Therefore, the temporal aspect cannot be overlooked, when analyzing a tech tree. By internal balancing we mean a process of modifying a tech tree for making it behave internally in a more favourable, balanced manner. If two technologies are supposed to be relatively on the same requirement level in the tech tree (e.g., they could belong to the same historical period), more or less same amount of efforts ought to be required for achieving them.

Typically, such efforts manifest themselves as gathering and spending *resources*. Time can be a resource type itself, but also cumulating assets, like *science points* or *energy*, can be used. In a more general characterization, of course, also noncumulating assets besides time can be allowed. However, in practice, such are seldom (if ever) used.

If a player is free to develop a technology, the time consumption for obtaining it depends often quite directly on the other resource requirements and the resource income at the corresponding game situation; a certain amount of resources must be accumulated and allocated for the development task in order to acquire the desired tech. Some additional time can be spent in carrying out the development task itself, in particular, if all the needed resources must be collected in advance.

The means and methods of gaining resources vary. Resources can be, for example, won in war, collected by peasants from mines, produced in factories, or gained by research work. However, the resource income can often be estimated as a function of game time, regardless of the exact earning method. When proceeding in a balanced tech tree, the requirements for developing each technology should be in accordance with the desired temporal flow (or "resource

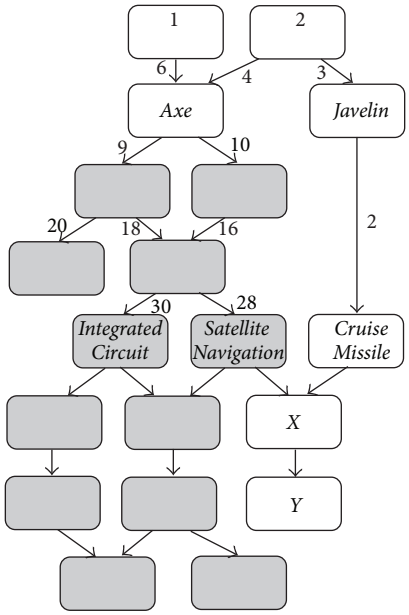

Figure 1: A problematic technology tree. Rounded rectangles depict technologies. They are of Boolean nature: either one possesses a technology, or not. The dependencies (marked with arrows) are of type OR. The numbers indicate costs in *gold nuggets* for advancing in the tech tree along the arrows. The costs for obtaining *entrance nodes* (ENs, the nodes representing the technologies that are initially available for development) are marked inside the corresponding rectangles.

flow") so the game proceeds smoothly and is gated as intended.

Consider the illustrative tech tree of Figure 1. Both *Axe* and *Javelin* are meant to be easily achievable primitive technologies. The minimum costs for obtaining them are of the same magnitude (six and five gold nuggets, resp.). Also *Integrated Circuit*, *Satellite Navigation*, and *Cruise Missile* are intended to be mutually "on the same level": they represent rather advanced technologies of the same historical era. All the three have powerful game effects, although this is not directly visible in the figure. However, via *Javelin*, *Cruise Missile* can be developed by using only seven gold nuggets. On the other hand, the minimum cost for getting *Integrated Circuit* is 62 nuggets, and that of *Satellite Navigation* is 60 nuggets. These two costs are relatively close to each other, as they should be, but the cost for obtaining *Cruise Missile* is significantly lower. Such magnitude difference implicates that the design is faulty. In this case, the problem is, naturally, the overly inexpensive technology *Cruise Missile*, which would, based on its achievability, belong to the same technological level with *Axe* and *Javelin*, but thematically, and based on the effects of the techs, this does not make much sense.

The problem with the design is also a flow problem; one should not be able to achieve an advanced technology like *Cruise Missile* too early; not only would such a feature be narratively awkward, but it might also manifest itself as a problem of strong paths, clearly superior routes to follow. Given the chance to obtain a powerful technology at an early stage via a certain route, probably that route is practically always chosen. The strong path problem can be alleviated by good internal balancing. If different techs are supposed to represent the same requirement level, the corresponding costs for obtaining them should be close to each other. Hence, a more balanced tech tree (in this sense) can be obtained by smoothing out differences between the costs of such techs.

In a balanced tech tree, the problem may only arise if the original assumption of the suitable requirement level for a tech is wrong. Looking from another perspective, effects of a tech should always be suitable for and on a par with the intended requirement level. Therefore, adjusting a tech tree in order to fix the problem requires estimates for game impacts of technologies involved. These estimates should be based on proposed requirement levels and offered game-dependent benefits.

So, internal balancing deals with a single tech tree, trying to make it better. By balancing between different tech trees, on the other hand, we mean a process aiming to make different tech trees behave similarly enough. This kind of balancing is important, for instance, when creating various tech trees to be used by different factions (e.g., nations) competing against each other. The common reason to use individual tech trees for the factions is the desire to clearly distinguish between them. Therefore, the dissimilarities between the tech trees should not be only cosmetic (e.g., different tech names): there should be actual variation in advantages, disadvantages, options, and meaningful strategies. However, the feeling of fairness in a game is important [13], and normally all the playable factions should be able to win. The playing should be enjoyable and meaningful. Therefore, none of the tech trees ought to be strongly underpowered or overpowered despite their differences.

4. Indicators for Characterizing Technology Trees

To make it easier to discuss measuring tech tree features, let us define some quality (and other) indicators. Let T be a tech tree with n distinct technologies. For simplicity, we assume here that there is only one resource type of interest, r, to be considered at a time. If there are several resource types affecting the situation, the indicator values can be defined separately with respect to each of them.

Useful global indicators describing the properties of T as a whole are at least

(i) its *size*, which we consider in terms of distinct technologies (and not the number of edges, as is the graph-theoretical convention), so here it would be n techs,

(ii) *tree requirement* (TR), the minimum amount of resources of type r needed to get all the n distinct technologies in T developed,

(iii) *average resource consumption for acquiring technologies* (ARCFAT),

(iv) *expected final tech coverage* (ETC),

(v) *average branching factor* (ABF) of T.

In order to define ARCFAT, a measure of local nature is needed. We call it *resources expected to be needed for acquiring a tech locally* (REAL). ARCFAT is the arithmetic mean of the REAL values of all the techs in T.

The REAL value concerning the development of technology t is the arithmetic mean of its local costs (in r). These costs correspond to the parent combinations that can allow (and are here assumed to allow) the development. In a simple (deterministic) case, a tech has only a single fixed local cost value, and the averaging over alternative parent combinations leading to the tech is unnecessary. In a more complicated case, the REAL value could be affected by, for example, randomization.

An interesting special case is REAL in respect to time (that we consider here as a resource among others). In this case, the REAL value for technology t is effectively the average of the expected development time requirements corresponding to the possible parent technology combinations allowing t to be developed. Such a measure can be straightforwardly used, for instance, to facilitate planning and constructing a tech tree, when targeting for some desired (expected) game duration.

The ETC value for T in a game featuring sequential technology development and technologies of Boolean development statuses can be defined as $e/(m \cdot n)$, where e is the expected gain of resource r of a game and m is the ARCFAT value of T. The ETC values are useful, when striving to tackle the problem of too limited possibilities to explore the tech tree during a game, or more generally, when trying to provide the players with the possibility to explore as much of a tree as desired in a typical game.

Knowing the exact ABF of a tech tree is usually not needed in time or space complexity analyses of the algorithms manipulating tech trees due to their modest sizes and the fact that the ABFs are typically small. However, one can draw conclusions on the structure and nature of a tech tree based on the ABF value of it. If the value is high, one probably has to really keep making choices frequently, but in the case of a low value, the tech tree probably has almost linear paths, and the focus is more on selecting the path(s) to follow and changing it when appropriate.

The values of the global indicator measures can be used for rough comparisons between tech trees. They are also useful in improving a single tech tree or designing such (and related game properties) from scratch. To clarify this, let us assume that we are creating a tech tree T and have formed a prototype with the following known (measurable) indicator values: ARCFAT = 350 r, TR = 5000, and size = 20 techs. We want ETC to be around 90 percent. Resources of type r are somehow produced or gathered during the game, and we want to define the details for the resource system. A simple estimate for the total amount of resource type r that one should be able to gain during a game can be obtained simply by taking the TR value and multiplying it by the desired ETC. In this case, the estimate is $0.9 \cdot 5000\,r = 4500\,r$.

Typically, the first technologies to obtain are relatively inexpensive, and the costs increase when advancing in a tech tree. Hence, technologies which are eventually not obtained in a game probably include many of the most expensive ones. Therefore, the estimated value might be more than enough. However, generally using the ARCFAT for calculating the lower bound in order to define the desired resource gain is a better solution. In our example, there are 20 techs, and on average each of them requires $350\,r$ to be developed. Hence, an ARCFAT-induced lower bound in this case would be $0.9 \cdot (20 \cdot 350\,r) = 6300\,r$ (during a typical game).

Often, it is a good idea to give players some extra resources to let them have more freedom to proceed via nonoptimal routes; all the players are not interested in optimizing exactly, when playing. Too scarce resource supply may lead to players never proceeding to expensive techs, thus rendering them unnecessary. On the other hand, of course, if there is an upper tech coverage limit defined that is not to be exceeded, resources should not be granted too generously, or the number of achievable techs should be limited in another way.

Consider another example. Let T and U be tech trees intended to be used for the same purpose by two different mutually competing factions (one tech tree for each of them). If the expected curves of income for these factions (in r, as a function of wall time or play turns) are similar, one can assume that the total amounts of r gained by the factions during a game are near to each other, as long as the game starts and ends at the same time for both players. Typically, the sizes of T and U should be about the same. This, of course, assumes that ARCFAT values and the desired ETC values are also near to each other. In some rare cases, this assumption might not hold, but nevertheless, the measures are useful and help in making desired adjustments. The ARCFAT values should be adjusted suitably for maximizing the enjoyment of advancing in a tech tree. It is frustrating to have to wait for a very long time for even simple technologies, but on the other hand, a game most probably has also other contents besides making tech choices, so all the gaming time cannot be used for that. To summarize, as a rule of thumb, the sizes and the ARCFAT values of T and U ought to be similar. If this is not the case, adjustments to at least one of the tech trees are probably needed.

A simple, but quite powerful, way to roughly adjust the global properties of a tech tree is to choose a suitable coefficient, c, and then, for each tech t, simply to multiply the needed resource requirements by c (for all the possible parent technology combinations allowing the development of t). This affects directly the REAL values and thus also ARCFAT measures. Moreover, via ARCFAT, the effect of the operation reaches to ETC. Game length can also be manipulated this way via REAL and ARCFAT values for time resource.

5. Indicators for Characterizing Technologies

Besides REAL, there are also other important indicator measures of a local nature to be found. Let t be a technology in a tech tree T with a total number of n distinct technologies, and let Q be the set of those technologies in T, into which there are nontrivial paths from t. Interesting local indicators for the single node t include at least

(i) *gating indicator number 1* (GIN1) = $a/(n-1)$, where a is the cardinality of the set Q,

(ii) *gating indicator number 2* (GIN2) = $b/(n-1)$, in which b is the number of technologies in Q that require t to be developed prior to their own development (i.e., there are no alternative routes leading to them and bypassing t),

(iii) *optimal cost* (OC) = the minimum amount of resources of type r needed to get t developed without any prior development in T,

(iv) *requirement ratio* (RR) = OC/TR (assuming TR differs from zero).

These local indicators can be used, for instance, to estimate which technologies should receive extra attention. As an example, a high value of GIN2 for a technology t indicates that it should be possible, maybe even easy, to obtain t, because otherwise a large portion of the existing tech tree is effectively not needed for anything.

The tech tree of Figure 1 has 19 nodes in total, so that $n - 1 = 18$. From node *Axe* 14 nodes (the grey ones plus X and Y) can be reached. Thus, GIN1 for *Axe* is $14/18 = 7/9$. The value is relatively high, as expected: it is easy to check visually from the figure that only a small portion of the tech tree is inaccessible via *Axe*. Technologies X and Y could be developed without *Axe* via *Javelin* and *Cruise Missile*. Therefore, these two technologies are not counted to number b when calculating GIN2 value. The 12 grey nodes in Figure 1 represent technologies that require *Axe* prior to being developed themselves. Hence, the indicator GIN2 has the value $12/18 = 2/3$, which is less than the value of GIN1 but still rather large; the grey nodes alone make a large portion of the tech tree, and *Axe* is a mandatory node in all the possible paths leading to them.

OC values were already calculated for the argument of *Cruise Missile* being too cheap to achieve in the example of Figure 1. For instance, the OC value for *Satellite Navigation* is 60 gold nuggets by minimization over alternative routes leading to its development. In order to demonstrate TR and RR, let us use even smaller example tree illustrated in Figure 2.

Let us determine the TR (with respect to r) assuming OR-type dependencies. This problem resembles closely the graph-theoretical minimum directed spanning tree problem, but with tech trees instead of one root there can be several ENs. To obtain A, B, C, D, E, G, and I, there are no choices to be made: the cost for getting these nodes developed is $(1 + 2 + 5 + 8 + 3 + 10 + 24)\ r = 53\,r$. We can consider them to have been paid for and developed. Now, the optimal choice for developing F is via D using $7\,r$, and for developing H (after developing F) it is via F using $15\,r$, so that TR = 53 + 7 + 15 = 75. Now, the OC value (still with respect to r) for, say, F, is 1 + 5 + 9 = 15, so RR value for F is 15/75 = 1/5. On the other hand, the RR value of I equals (1+8+10+24)/75 = 43/75. This value is considerably larger than 1/5, which indicates that I is more expensive than F. Therefore, I is suitable to be the more advanced technology of the two.

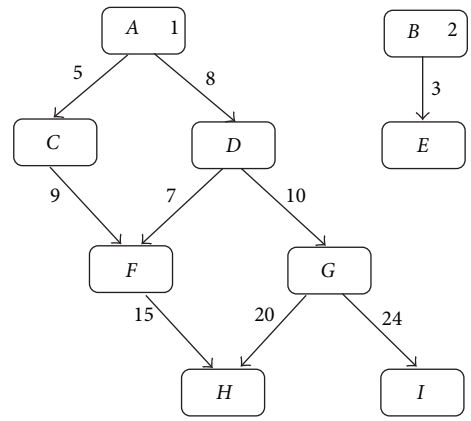

Figure 2: A simple example tech tree depicted using the conventions established in Figure 1. The costs are expressed in resources of type r.

6. Measuring the Indicators: Implementation

In a previous work [2], we proposed a generic approach for implementing technology trees to obtain diverse benefits. We also introduced a software tool for creating technology trees easily and partially automating the process of tech tree creation. We have now augmented our software tool, TTT, with a capability of measuring important tech tree properties.

Our previous paper [14] discusses temporal layer analysis. The basic idea is presented in Figure 3. A technology tree is converted into a forest consisting of real (graph-theoretical) trees by duplicating nodes (representing technologies) as necessary. Then, the nodes, having visual representations, are arranged topologically in a way that makes it easy to analyze tech tree properties visually and find problems. In Figure 3, there are six different time layers corresponding to specific points in time, and there is a node corresponding to a technology on a layer if and only if it is possible to get the tech developed at the time represented by the layer. Time is assumed to be the only resource type in use, the development status of each node is Boolean, and having one developed parent is a sufficient precondition for developing a tech node in this setting (that is, the dependencies between technologies are treated as OR-type ones). The progress is assumed to be sequential and without any slack time.

We implemented a modified version of this conversion in TTT. The idea is still mostly the same, but in order to limit the amount of nodes to be generated and drawn, we only consider possible paths leading to a given technology, not necessarily the whole tech tree. On the other hand, the topologies are presented for all required resource types, not only time. We call this approach *generalized time layer approach* (GTLA).

Resource demands (of the resource types under scrutiny) are accumulated into each node along different routes starting from the ENs. The resource requirements of each node are resolved for each requested resource type, and the values are "pushed" top-down to (new) nodes corresponding to the children in the original tech tree. Then, these newly created nodes are processed similarly, and the process continues, until all the nodes have been exhausted.

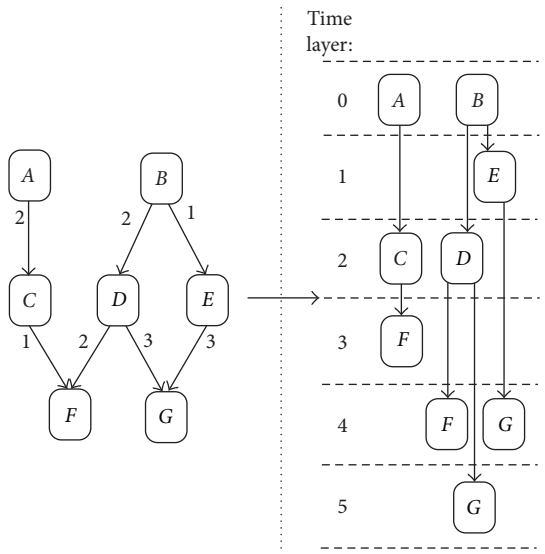

Figure 3: Generating a forest (on the right-hand side) based on a tech tree (on the left-hand side) and representing the optimal times to achieve technologies via different routes topologically. The numbers next to the tech tree edges are corresponding time requirements.

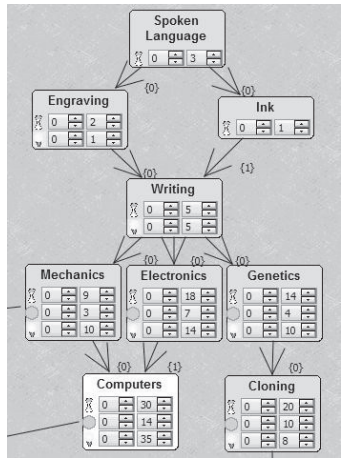

Figure 4: A part of a simple example tech tree. A partial screen capture from TTT.

Consider the tech tree structure of Figure 4 and applying GTLA to obtain possible paths leading to *Computers*. In Figure 5, the graphical presentation of these paths (in terms of different resource type requirements) is shown, as presented in the TTT UI.

From the graphical representation, it is easy to detect, for instance, the paths leading to optimal total resource consumptions in terms of the resource type of interest as well as the OC values themselves. Even simpler representation, with fewer plot points to consider, can be obtained by aggregating different resource types into a more general resource requirement plot. Different weights for distinct resource types can be used.

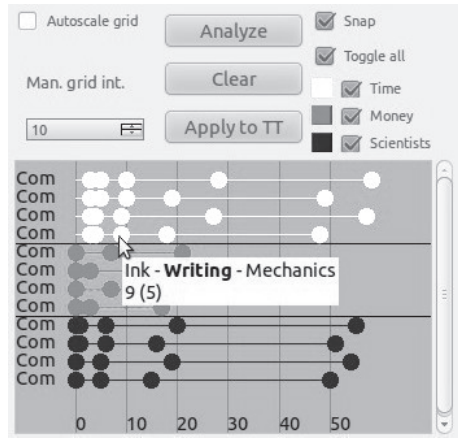

Figure 5: Possible technology paths to *Computers* in the tech tree of Figure 4 plotted for three resource types. The paths are depicted as horizontal successions (from left to right) of circles, each representing a technology. A partial screen capture from TTT.

So, GTLA can be used, among other things, to determine OC values. However, the method in its simplicity is only suitable for tech trees, in which any technology can be developed, if allowed by any parent technology. This is (typically) true for tree-like tech trees, in which a technology has at most one parent, and for tech trees based on OR-type dependency relations, like the one in Figure 4. In the example tree there are two technologies, *Writing* and *Computers*, with two parents each. However, having a single parent technology developed suffices to develop the corresponding child. In the TTT UI (see Figure 4), this fact is visible as the identifier sets attached to edges starting on different parents, {0} and {1}, containing different elements.

The idea is that the identifiers, *and-bunch identifiers* (ABIs), of the edges arriving from the parent nodes determine sufficient edge sets that allow a tech to be developed. In order to develop it, for an ABI present in the ABI set of some incoming edge, it must hold that all the incoming edges having this ABI in their ABI sets must *fire* (accept the development) simultaneously [2]. In this paper, we assume for simplicity that an edge fires, whenever the technology corresponding to its start node has been developed. This is typically the case with real-world tech trees with binary tech development statuses.

In order to illustrate the idea behind ABIs, let us modify the example tech tree of Figure 4 a bit. In the tech tree of Figure 6, there are AND-type dependencies involved. *Spoken Language* is not anymore a prerequisite for *Engraving* or *Ink*, and to be able to develop *Writing*, one must have in addition to *Spoken Language* either *Engraving* or *Ink* developed. The freedom to determine several sufficient AND-type parent groups for a technology (and the ABI set characterization) adds considerably to the overall flexibility of tech trees and facilitates using thematically sensible dependency structures.

GTLA proceeds locally from ENs towards the leaf techs without considering all the necessary requirements (parents) when used with this tech tree. Hence, it gives too optimistic

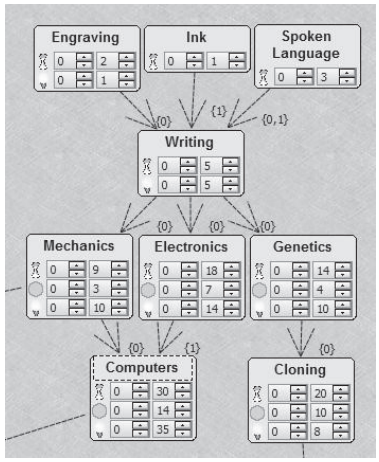

Figure 6: A modified example tech tree. A partial screen capture from TTT.

```
(1) function approximate_OCs():
(2)     C ← the set of ENs
(3)     while |C| > 0 do
(4)         N ← ∅
(5)         for each n in C do
(6)             handle_node(n)
(7)         swap(C, N)
```

Algorithm 1: A function for going through a tech tree appropriately. Executing this results in having OC estimates for each node, as well as estimated optimal ancestor sets and optimal ABIs.

```
(1) function handle_node(n):
(2)     n.counter ← 0
(3)     for each a in n.abi_sets do
(4)         R ← ∅
(5)         abi_solved ← true
(6)         for each incoming edge e of n with ABI a do
(7)             p ← e.start_node
(8)             if p.solved then
(9)                 R ← R ∪ {p}
(10)            else
(11)                abi_solved ← false
(12)        if abi_solved then
(13)            n.counter ← n.counter + 1
(14)            A ← ⋃_{p∈R} p.opt_anc
(15)            s ← n.cost(a) + Σ_{v∈A} v.cost(v.opt_abi)
(16)            if s < n.opt_cost then
(17)                n.opt_cost ← s
(18)                n.opt_abi ← a
(19)                n.opt_anc ← A ∪ {n}
(20)    if n.counter = |n.abi_sets| then
(21)        n.solved ← true
(22)        for each c in n.children do
(23)            if not c.solved then
(24)                N ← N ∪ {c}
(25)    else
(26)        N ← N ∪ {n}
(27)        for each p in n.parents do
(28)            if not p.solved then
(29)                N ← N ∪ {p}
```

Algorithm 2: A function used by approximate_OCs for handling an individual node and making the necessary additions to the next node layer.

OC results in this case. Therefore, a more general method for coping with this kind of more complex tech trees and estimating their OC values and optimal routes was developed and implemented in TTT. Now, each tech may have several different parent subsets allowing its development, and each such subset may have its own respective resource requirements. In other words, a technology may have several alternative price tags, with the actual cost to be paid depending on the developed prerequisite tech combination. The OC estimation algorithm, usable with acyclic tech trees, is presented in Algorithm 1 as the pseudocode function approximate_OCs. The function handle_node, used by approximate_OCs, is given in Algorithm 2.

The functions do not aim at plotting the possible paths leading to a tech, since their number in nontrivial tech trees can be large because of different possible orders of developing prerequisites. Instead, for every technology in the tech tree, the OC value is estimated and a corresponding ABI set, via which one probably achieves the tech optimally, is determined. Estimated optimal route for developing the technology of interest is available in the form of the set of optimal ancestors (*opt_anc*) for each node after executing approximate_OCs.

To simplify the presentation in Algorithms 1 and 2, the resource type under scrutiny has been omitted in the pseudocode, and it is assumed that all the relevant operations and variables are with respect to it. The functions could, of course, take resource type parameters and use them in bookkeeping.

The basic idea is to start with the ENs as the active set (layer) C of nodes to be handled. For each node in C, the function handle_node is executed. It forms the layer N to be handled during the next round, and the execution proceeds this way a layer after another, until the values have been solved for the whole tech tree. The algorithm stops, because eventually all the nodes in a tech tree of a finite size will be solved, and thus N will be the empty set after the loop of the lines (5) and (6) of approximate_OCs. Each call to handle_node either marks a node solved and adds its children to N or adds its unsolved parents to N, which prevents looping infinitely.

The obtained values are only estimates, because *v.opt_abi* (handle_node, line (15)) is not necessarily the optimal ABI with respect to n but only v itself. However, in practice, the estimates obtained this way for real tech trees are typically accurate, and making the simplifying assumption of global optimality of the determined optimal ABIs reduces the required computing time and space usage into a sensible level. The optimal ABI values can be used for determining the costs

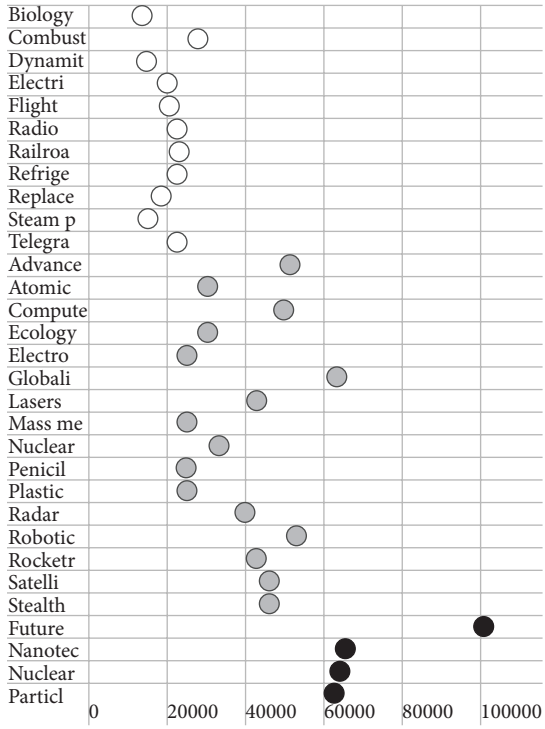

Figure 7: Minimum science requirements for the technologies of the last three eras in Civ 5, as plotted by TTT. (In order to improve the image quality for publication, the figure was redrawn based on a TTT screen capture.) Distinct techs are represented on their corresponding rows, and the horizontal locations of circles (their midpoints) represent the OC values of the corresponding techs. OC values increase linearly along the horizontal axis from left to right, as indicated.

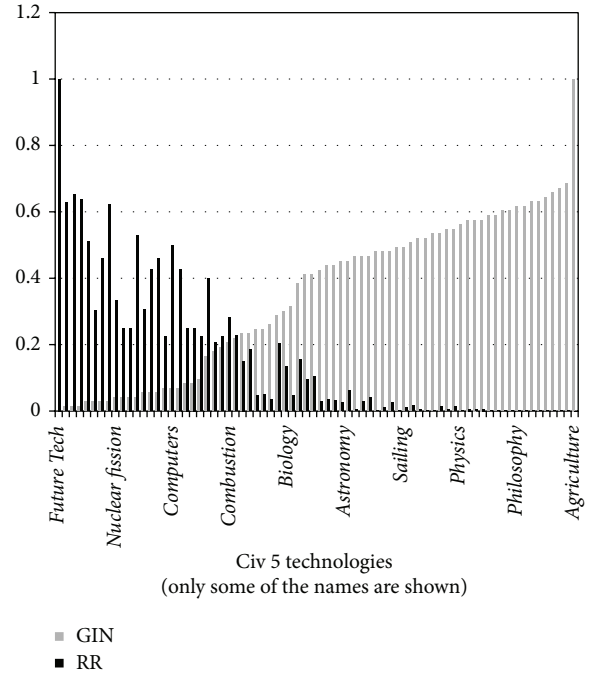

Figure 8: GIN (= GIN1 = GIN2) values and corresponding RR values of the technologies in the Civ 5 tech tree.

for individual techs, when estimating the TR value for a tech tree.

7. Measuring a Real-World Tech Tree

To test our tool improvements, we created a TTT representation of the tech tree of Civ 5, because it is a good representative of technology trees used in contemporary digital games. The game is rather recent, but the technology tree is used in a conventional fashion. The Vanilla version with *science* cost estimates taken from a web source [15] was used. We ran the algorithm of Algorithm 1 for the tech tree. Because of the structure of it, the exact OC values were obtained. The operation took time of 16 milliseconds with a standard desktop computer (Intel Core i5-3470 running at 3.20 GHz, 16 GB RAM, 64-bit Microsoft Windows Enterprise).

In Figure 7, the *science point* (or "*beaker*") OC results have been plotted for the technologies of the last three eras of the game: Industrial Era (white), Modern Era (grey), and Future Era (black). These kinds of plots are beneficial for checking visually that technologies are as easy or hard to obtain as they should be or spotting problems concerning internal balancing. As the eras in Civ 5 represent distinct intervals in the (temporal) continuum of overall technological development, it would make sense for technologies categorized to belong to the same era to have minimum science requirements of the same magnitude.

As can be seen in Figure 7, the requirements for the Industrial Era technologies do not differ much from each other. Modern Era has more variance in this sense, and the technology *Future Tech* of the Future Era seems to be clearly an outlier that should be checked carefully, if the technology tree was under development. In this case, however, the huge amount of science points required is due to the unique nature of *Future Tech*; it can only be developed after developing all the other techs.

Besides OCs, we let TTT also compute other interesting (local and global) indicator values for the technologies of the Civ 5 tech tree. There are no surprises in the results. GIN1 values are equal to corresponding GIN2 values, because Civ 5 tech tree does not offer alternative routes to achieve technologies. In Figure 8, the technologies are ordered into an increasing order based on their GIN1 values (= GIN2 values), marked simply as GIN in the figure. Also, the corresponding RR values are shown in the same figure.

When GIN values increase, RR values tend to decrease. This makes sense, because near ENs there are (typically, and not only in this case) inexpensive technologies, via which one has to proceed in order to access the other parts of the tech tree. On the other hand, the final technologies typically require lots of resources and they limit access to only few even more advanced techs.

The TR value for beakers is 100,487, the ARCFAT value is 1,357.93, and the ABF is approximately 1.49. TR and ARCFAT do not tell very much in a case of a single tech tree without

anything to compare to, but based on the modest ABF value and the additional fact that the tech tree is connected, one can conclude that it is also rather deep.

It is worth highlighting that the indicator values obtained might be rather different, if measured from a patched version of the game (possibly augmented with downloadable content packages), as the tech tree properties have changed since the Vanilla version. The fact that the tech tree has been modified several times demonstrates that it is really an important part of the game. As mentioned, used costs are also only approximations; the exact in-game costs depend, for instance, on the number of cities the player has.

8. On Correcting a Tech Tree

Whenever measurements indicate problems, taking corrective steps can be either easy or tedious. With TTT, adjusting the tech tree structure and modifying local properties, like resource requirements and dependency relations of a single technology, are easy, since the tool has been created for effortless technology tree manipulation. The novel features make such manual adjustments even easier. Especially worth mentioning is the fact that the GTLA view (see Figure 5) allows (imposing necessary restrictions) the user to drag technologies along their respective paths and to commit the corresponding resource requirement changes into the actual tech tree presentation of the program, from which functional technology tree code is generated automatically, when desired. This way the user can see the effects of planned changes to other technologies visually before actually applying them.

As far as global adjustments (affecting the characteristics of a tech tree as a whole) are considered, our tool so far supports setting the desired TR value, based on which the system is capable of adjusting the tech tree multiplicatively. The modification is performed simply by determining and applying a suitable multiplier for all the technology costs in the tech tree.

9. Conclusion

In this paper, we have introduced indicator values and discussed algorithms and our implementation for analyzing technology tree features for proper adjustments. The implementation was also tested with a real, popular computer game, and thus its capability to produce and visualize data, which we strongly believe to be useful, was verified.

As future work, more general, important, and measurable tech tree features should be pointed out, and corresponding measuring and correcting procedures ought to be implemented. Moreover, the automated analysis features currently present in our software should be improved. Also, a considerable number of real-world tech trees ought to be analyzed in order to find good practices and typical tendencies to guide in the further development and fine-tuning of adjustment automation procedures.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

References

[1] N. Hallford and J. Hallford, *Swords & Circuitry: A Designer's Guide to Computer Role-Playing Games*, Stacy L. Hiquet, 2001.

[2] T. J. Heinimäki and T. Elomaa, "Facilitating technology forestry: software tool support for creating functional technology trees," in *Proceedings of the 3rd International Conference on Innovative Computing Technology (INTECH '13)*, pp. 510–519, London, UK, August 2013.

[3] T. Ghys, "Technology trees: freedom and determinism in historical strategy games," *Game Studies*, vol. 12, no. 1, 2012.

[4] P. Tozour, "Introduction to Bayesian networks and reasoning under uncertainty," in *AI Game Programming Wisdom*, S. Rabin, Ed., pp. 345–357, Charles River Media, 2002.

[5] E. Schaefer, "Blizzard entertainment's Diablo II," in *Postmortems from Game Developer*, A. Grossman, Ed., pp. 79–90, CMP Books, CMP Media LLC, San Francisco, Calif, USA, 2003.

[6] D. Morris and L. Hartas, *Strategy Games*, Ilex Press Ltd, Lewes, UK, 2004.

[7] M. Barton, *Dungeons and Desktops: The History of Computer Role-Playing Games*, CRC Press, 2008.

[8] T. J. Heinimäki, "Technology trees in digital gaming," in *Proceedings of the 16th International Academic MindTrek Conference (AMT '12)*, pp. 27–34, October 2012.

[9] T. Owens, "Modding the history of science: values at play in modder discussions of Sid Meier's CIVILIZATION," *Simulation & Gaming*, vol. 42, no. 4, pp. 481–495, 2010.

[10] R. Rouse III, *Game Design: Theory and Practice*, Wordware Publishing, 2nd edition, 2005.

[11] A. Gazzard, "Unlocking the gameworld: the rewards of space and time in videogames," *Game Studies*, vol. 11, no. 1, 2011.

[12] J. Huizinga, *Homo Ludens—A Study of the Play-Element in Culture*, Beacon Press, 1971.

[13] E. Adams, *Fundamentals of Game Design*, New Riders Publishing, 2nd edition, 2009.

[14] T. J. Heinimäki, "Considerations on measuring technology tree features," in *Proceedings of the 4th Computer Science and Electronic Engineering Conference (CEEC '12)*, pp. 145–148, Colchester, UK, September 2012.

[15] List of technologies in Civ5, 2014, http://civilization.wikia.com/wiki/Technologies%28Civ5%29.