Composition Parsing: Office Space Planning and Automation as Translator

CHARLES DRIESLER
Pratt Institute

The ability to synthesize design intent with computable values generates a novel method for composing space. An intermediary automated layer, between designer and their computational tools, is proposed and implemented for the space planning problem of the office test fit. This system for composition parsing, when given a combination of explicit or gestural inputs, will reduce them to a collection of small and simple problems. A reductive approach allows the system to deliver computationally easy requests to existing systems, interpret the results, and reconstruct a comprehensible output for the user. By focusing on the ability to translate disjoint compositional understandings of a given space, the system does not need to enforce a certain geometric construction of the problem or prepare layers of contingency for harder cases. It must only maintain the ability to translate between the people and systems that do. The paper opens with an investigation of how recently published algorithms interpret the composition of their given space planning target and how their implementations approach the question of user interactivity. It then outlines the three phases of the proposed system before concluding with an evaluation of its results and limitations. The paper ends with speculation on necessary steps for the next generation of this strategy and on future forms of user intervention with otherwise fully automated results.

INTRODUCTION: OFFICE SPACE PLANNING

The architectural computation arena has recently renewed its interest in the space planning problem of the office interior test fit: the maximal placement of bodies and furniture given an empty floor plan and a client's programmatic needs. The problem is a clear target given the relative low resolution of its solution space when compared to larger *tabula rasa* architectural projects. The test fit's slate, while empty, is not blank; its constraints (floor area, geometry, site, orientation, programming requirements) are almost totally inherited, and their common parameters (desk size, placement affinities, population configurations) are necessarily tested only within a small range of acceptable values. In short, the office test fit is a popular target because it seems that it can be simplified for problem areas automation is good at: geometric and numerical relationships. Many of the latest advancements then frame their success criteria around questions like *"How many more desks were we able to fit than other manual designs?"* and *"What is the strength of this configuration when considering daylighting, adjacency, and density?"* or any other number of performance metrics. Consistently, projects are evaluated based on output provided by automation without imposing or allowing for an interpretation of the problem by the designer. Said another way, these systems play a game based on their specific understanding of the space planning problem and reduce the designer to a bookend: someone to input values at the start and to select a result at the end.

This comes at a moment when the industry is reconsidering its relationship with technology.[1] In divorcing the designer from the processes of designing, each tool is also restricting its relevance to a subset of office types; the character, or *composition,* of the problems solved must be similar (and, often, rectangular). To allow for more robust systems and to enable a greater level of interactivity between designer and automation, this research proposes splitting the current popular methodology into two phases. This first step is an initial composition parsing of the space planning problem as-proposed. The process "reads" given constraints and translates an interpretation of the floor plan and program requirements into a collection of smaller geometric problems. Then, second, the system will match these smaller problems to both novel and existing population strategies in the same camp as those mentioned above. The network relationships established in the first phase are maintained as the smaller problems are solved in isolation. This implementation is an engine for the persistent translation of automated compositional understanding. It negotiates shortcomings of purely geometric population strategies by simultaneously maintaining design intent and converting into simple relationships between numbers and shapes. As an intermediary layer between designer and computation, the system promises a form of automated design that can build on the many strengths of existing algorithms *and* allow for user intervention when necessary.

DERIVING THE COMPOSITION PROBLEM

The following systems, while built around a divorced relationship with the designer, do so consciously and offer sophisticated results in return. To be clear, this paper *is not* a critique of the logic or methodology in other similar projects. Development began with an audit of limitations in black box style systems (that treat designers as choosers), though, and observations on the projects below acted as precedent for the proposed solutions. This project, in addition to generating useful output, aims to provide greater piecemeal interactivity between the user and the system. This sentiment is often also expressed by the papers cited. The Anderson paper's abstract

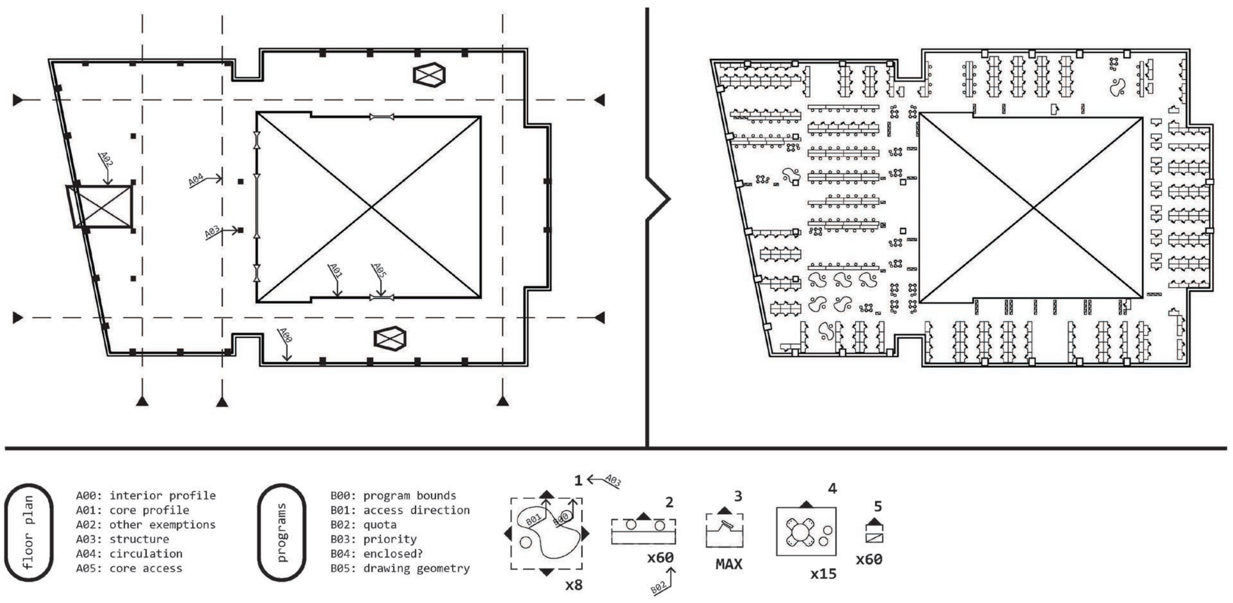

Figure 1. Required geometric input for both the floor plan and program requirements. The results of this configuration are also shown on the right.

specifically mentions that they hope for "a more dynamic and collaborative interaction between computer design software and human designers in the future."[2] Disclaimers aside, the next section evaluates three recently published systems for their degree of interactivity and suggests moments in their structure that may be responsible. Alternative strategies for both how composition can be translated and why it should be considered a computational problem are offered in response.

WeWork's 2018 desk layout generator will output several configurations of desks for a single given room. Multiple options are rapidly generated, and then the user is asked to make a final decision on which layout to implement. While the WeWork team developed a sophisticated model to evaluate the success of their program's result against several existing designs and criteria, a potential corollary for compositional understanding, they chose to defer to the user's final judgement. This is justified by a comparative study that found the system "matched the efficiency of the architects on 77% of offices, and achieved a higher efficiency 6% of the time." By other metrics, the system "achieved a 97% match rate," meaning it "completed this design task as well as a designer and in a shorter time."[3] After creating the opportunity to let machine preference supersede a human's, chooses to reinforce the designer's contemporary position as chooser. Further development after publication has prioritized deploying the algorithm to larger problems instead of reconfiguring this interaction paradigm, suggesting a belief that "just choosing" is a sufficiently useful form of interaction.

Autodesk's 2016 space plan generator similarly focuses on rapid option output, but it also immediately reduces the programmatic requirements to a pixelated geometric model of subdivision based on linear slices. This method is computationally useful, but it also restricts any solution's relevance to a subset of the problem space, even if a large portion of the logic is generally valid for the issue of space planning.[4] While a certain understanding of floor plan composition is enforced here, it is a clear example of how spatial configurations can be translated to geometry and is a primary inspiration for our own geometric implementation.

Another later attempt at automated space planning by Autodesk, Project Discover, tackled a more specific problem with a single site and static set of goals. They employed a genetic algorithm to effectively consider several competing design criteria and arrive at a sufficiently provable "best" option. They conclude, though, with a few caveats: the stochastic nature of their system means the solution is only optimal based on the limited input provided. Similarly, regarding scope, they argue that "some aspects, such as beauty, cannot be quantified, and thus need to be considered once the generative design process is complete."[5] Here, composition is thoroughly considered by the genetic model at the cost of a large degree of control by the designer. But a suspicion is also called out: if qualitative information cannot be objectively mapped to a quantity, where do we situate the machine? This project proposes approximation through translation, and the implementation follows below.

THREE PHASES OF COMPOSITION PARSING

This project's reaction to the compositional problem is implemented as a collection of custom components for Grasshopper, a visual programming interface for McNeel's modeling software Rhino. It does not attempt to develop a more intelligent version of any of the above algorithms.

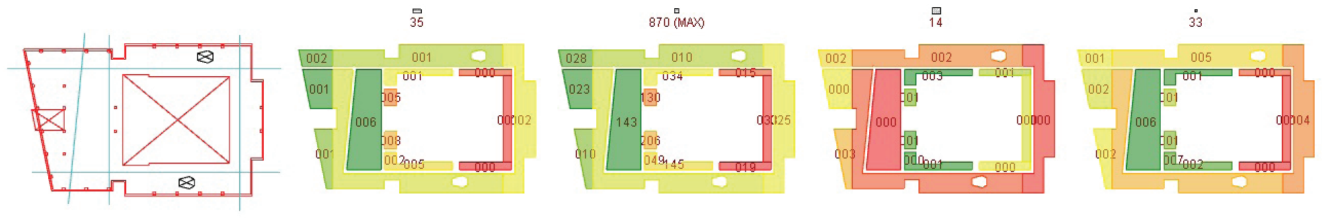

Figure 2. Neighborhood generation based on geometry at left. Program zone preference and quota distribution are superimposed at right, in priority order from left to right, where green is most preferred and red is least. Note that the lowest priority item has the least placement in the popular trapezoidal zone.

Instead, effort is focused on reducing a given problem to a collection of smaller ones that they are sufficiently capable of solving. (More specifically, a floor plan to a collection of rooms with egress.) The following section outlines the three primary phases of this process, internally called composition parsing, as well as the resultant collection of smaller packing problems. It concludes with a short note on how these results can be populated.

The system asks the user to declare a collection of curves and number values as input. These represent baseline constraints in floor plan geometry and programmatic goals. A desk, for example, is constructed from a rectangle for occupation bounds, the directions it can be accessed from, and a number for the minimum amount that must be placed. There is an immediate distinction, however, between objective and gestural inputs. Some information, like the floor slab profile, is unchangeable and treated as such by the system. But it also asks for other information with more flexibility in mind. When declaring desired circulation, the user is prompted for multiple axis that traverse the floor plan instead of an explicit procession represented as a line. Similarly, egress access to the core is input as regions that must remain unobstructed; the system will later decide how best to generate connections between these zones and primary circulation. This combination of input types gives the system enough freedom to interpret designer intent without forcing her to sacrifice precision or limit the solution space to a subset of geometries (figure 1).

The first phase of composition parsing reconstructs the floor plan as a handful of discrete neighborhood regions. It starts by generating a collection of ideal circulation configurations in a two-step process. First, the axes are fragmented at their intersections and categorized based on what they connect (e.g. core to perimeter, circulation to circulation). Second, several different heuristics reassemble the segments based on their adjacency to the perimeter, core, and other segments. These rule sets are derived from observations of existing projects, but a single "best" is not yet enforced. Each configuration is then connected to all necessary egress regions, and this collection of lines is used to slice the floor plate into patchwork zones. The likely irregular geometry is then further fragmented into its most rectangular component parts. Finally, each collection of circulation paths and zones are scored based on their "potential" for the set of program items requested. This is a simple check for correlation between program size and number against the size and proportion of generated zones (i.e. a request for several large programs will lead to a higher score for a configuration with more large zones). The highest scoring configuration is then passed along to the next phase. In this way, the system's first move is to reduce intent to a compositional read of the user's request that is computationally accessible: rectangular islands with simple topological relationships.

The geometric simplification of the first phase is necessary for the success of the second: distribution of program items to each neighborhood. In parallel with the first phase, the system interprets explicit program quotas with each item's attributes like geometry and privacy. Once the first phase completes, the system lists each neighborhood zone in order of "preference" for each program item. In another form of composition parsing, it will then consider both programmed heuristics ("offices cluster near the core") and user declared requirements ("this meeting table cannot sit against the perimeter"). A phase of horse trading follows, where each program "bargains" with the others for space in their most desired zones. Conflicts are resolved with a basic priority declared by the user for each program at the beginning of the process. The result of this second phase is an automated synthesis of floor plan composition and programmatic affinities (figure 2). Each algorithm responds to earlier generated geometric representations of user intent, preserving designer agency.

At the start of the third and final phase, the gestural curves and numbers input by the user have been translated into a handful of smaller zones. Each zone is also assigned a subset of the program requirements to handle. Before population begins, however, each zone must be further reduced to the simplest collection of packing problems possible and normalized in some way to allow a uniform application of placement rules. (Note that the desire for simplicity is outlined in the above discussion of other space planning algorithms. They work especially well for small rectangular spaces with a limited number of items.) The system achieves this standardization by slicing the previously rectangularized zones along

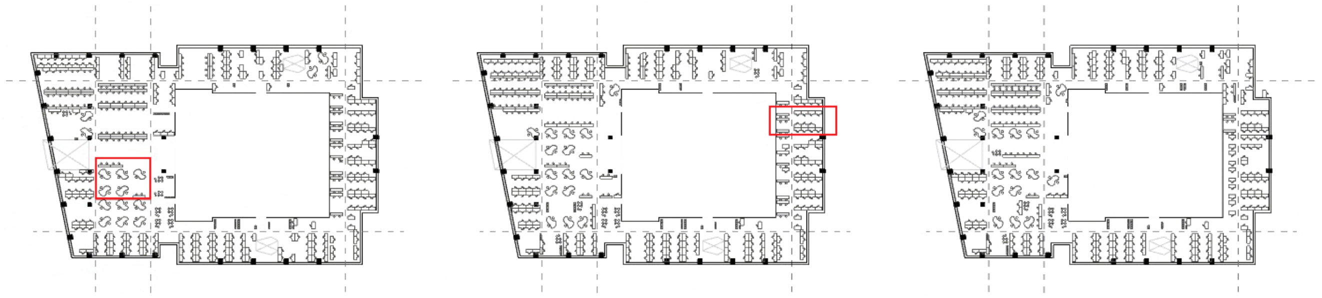

Figure 3. Iterative solutions with the same input configurations, from left to right. The user input in between each, noted as a red square, instructs the system to preserve that region of the solution. This region does not have to correlate with the substructure of neighborhoods and cells.

their narrow axis. The distance between each slice is determined by the sizes of the remaining items to assign. As each "cell" of the zone is created, this remaining quota total for the zone is reduced based on expected population results. At the end of the slicing process, each neighborhood zone has been separated into a collection of parallel cells (see second column of figure 4). A further subset of the program's quotas has also been assigned to each cell.

So, before population even begins, Desk Jockey has already decided the optimal size of each cell and the ability of each to fit an assigned quantity of program items. Placing each item is only necessary to confirm fidelity or to represent the result. Since the system consistently reduces the space planning problem to simple rectangles, it responds best to a basic "array along edge" population strategy. The nuanced aspects of more sophisticated algorithms (e.g. Anderson's perimeter-based population strategy) are not necessarily applicable, as the problem has been simplified to an easy case for each. For this reason, this research advocates for the computational *reduction* of problems, as opposed to computationally *robust* or *adaptive* algorithms. This ideology promises more successful inroads for the qualitative problems of design.

INTERACTIVITY

A second strength of this implementation, largely because of its reductive approach, is the possibility for greater user engagement with the design automation. Because the first phase begins with an interpretation of the floor plan, if it can also translate user intent into a compatible format for that interpretation, it can now also allow the user to provide more sophisticated instruction. One example of this in the current implementation is in the ability for the user to "maximize" a certain program request. If a zero is entered at the start, instead of the minimum requirement expected, it gets replaced by the maximum number of items that could possibly fit on the floor plan. The horse-trading algorithm in the second phase can negotiate the large request. The cell assignment process in the third phase will also understand that, even if it can't place every single item of the maximized program, its best attempt is what's being requested.

The system enables more robust types of interaction because it establishes its own language for the office test fit but *does not* enforce how it's used. The current implementation allows the user to, once a solution is generated, "lock" an arbitrarily large region and re-compute. The system sees this as a new request, with the region treated the same as a hole in the floor, and the program requirements are decreased based on how many already exist in the region (figure 3). The current language is sufficiently verbose to close the gap between what the user intends and what the algorithms can interpret, with some caveats explained below.

EVALUATION

Ultimately, the current implementation is not different in its relationship to the designer than other previously mentioned "bookend" algorithms. Each run is monolithic; the user's primary interactions are to deliver input and receive a total result. Its strength lies in its ability to obfuscate that relationship by interpreting interaction as "normal" input and by relating new solutions to previous ones. This provides an illusion of interaction that is sufficiently correlated with the user's expectations.

The potential interpretations of the floor plan are also still largely limited. The "nested reduction" system enforces a solely hierarchical type of solution. Relational data, like distance to other programs, can only be considered instantaneously on placement. And while evaluation of relational information or attributes of instantiated programs (e.g. approximate noise level of desk clusters, shading caused by perimeter offices) is possible, using that same information as criteria for placement is difficult. Still, the current version delivers consistent results for floor plan geometry that can be reduced to generally rectangular cells. This is not dependent on an overall orthogonal or rectangular exterior profile (figure 4).

This paper outlines the proof-of-concept phase for a more ambitious and robust project in development by HOK. All relevant code has been released under a free software license on the company's public repository.[6] Current development

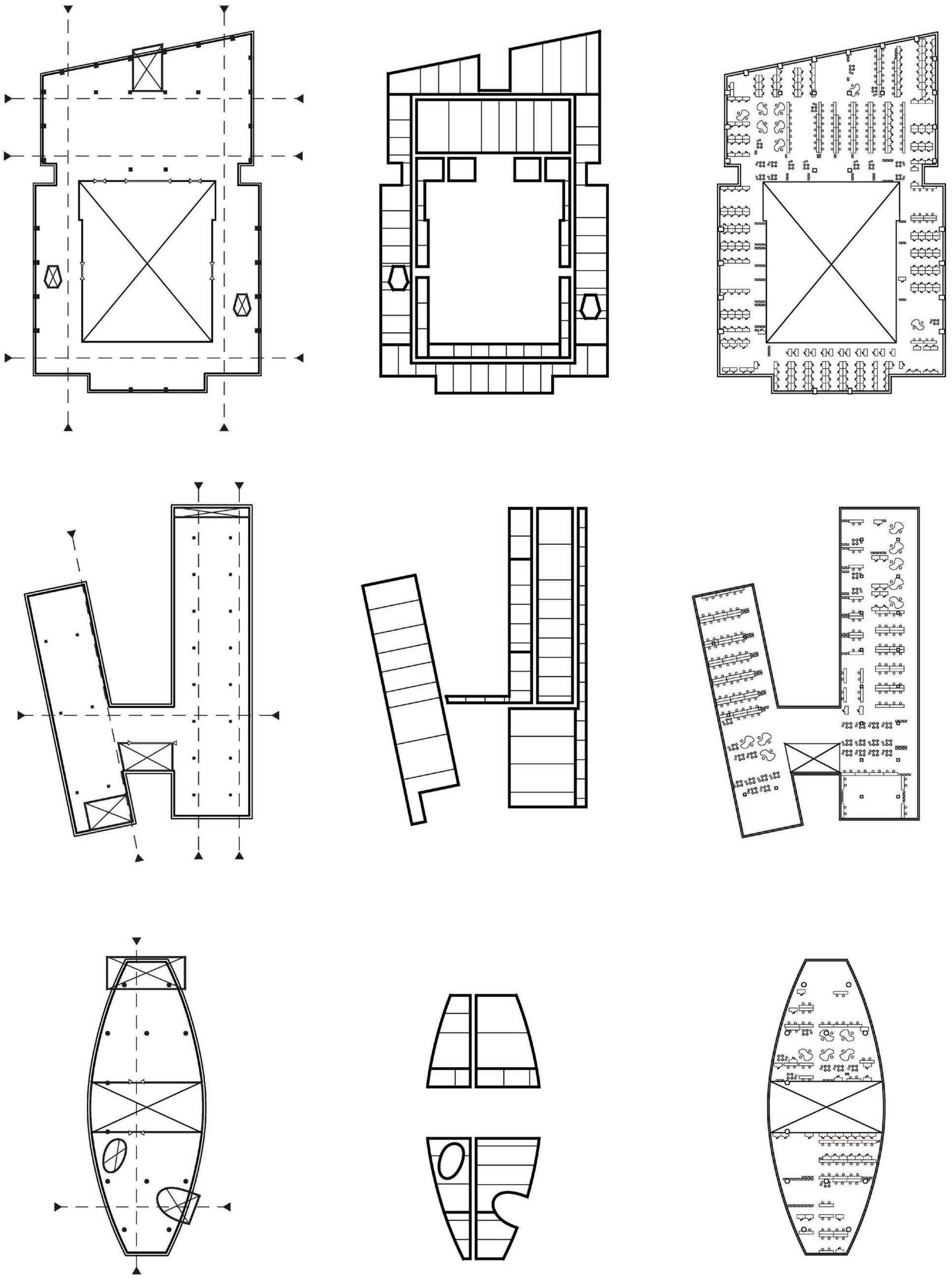

Figure 4. Results on three floor plans. Column information, from left to right: geometric input, result of composition parsing, result after population.

effort aims to compensate for these shortcomings by completely atomizing its component parts. Requiring a direct request for each type of functionality from the designer necessarily increases their control over the process and understanding of how it works. The team also plans to increase the amount of intelligence that can be stored within each program. Limiting their interpretation to a rectangle with an orientation makes sense for a simple packing problem. But more complicated relationships, necessary to solve contemporary schemes like the agile office (where one desk is not equivalent to one body), require more information to be attributed to each program.

CONCLUSION

The paper discussed the inherent restrictions in black box style approaches to the problem of automated space planning. Existing algorithms and implementations that address this problem were studied for their approaches to interactivity and compositional understanding of the office test fit. An intermediary computational layer was proposed. This system translates designer intent and space planning constraints into a consistent geometric language. With this language established, it can synthesize a combination of explicit and gestural inputs into a network of placement relationships that must be maintained. In parallel, it also reduces the geometry of a given floor plan to a collection of smaller, simpler, and generally rectangular packing problems. The population algorithms employed can then be much simpler, so long as the relationships are maintained, because the problems are simpler. This composition-heavy approach promises two primary benefits: a more generally applicable solution space because of the reductive attitude taken, and a means for better user interaction so long as intent can be translated into the algorithm's geometric language.

ENDNOTES

1 Carl Galioto, "Software That Should Be Unifying the AEC Industry Is Impeding Progress and Innovation," *Architecture Today* (September 2018).

2 Carl Anderson et al., "Augmented Space Panning: Using Procedural Generation To Automate Desk Layouts," *International Journal of Architectural Computing* 16, no. 2 (2018): 164–177.

3 Anderson et al., 164.

4 Subhajit Das et al., "Space Plan Generator: Rapid Generation & Evaluation of Floor Plan Design Options to Inform Decision Making" in *ACADIA // 2016 - Posthuman Frontiers: Data, Designers, and Cognitive Machines - Proceedings of the 36th Annual Conference of the Association for Computer Aided Design in Architecture* (Fargo, ND: ACADIA, 2016), 106-115.

5 Danil Nagy et al., "Project Discover: An Application of Generative Design for Architectural Space Planning," in *2017 Prooceedings of the Symposium on Simulation for Architecture & Urban Design* (San Diego: The Society for Modeling and Simuation International; Simulations Councils, Inc., 2017), 59-66.

6 GitHub, HOKGroup/burolandschafter, "Engine for Automated Space Planning (in development)." https://github.com/HOKGroup/burolandschafter.