Interactive Architectural Design with Diverse Solution Exploration

Supplementary Document

Glen Berseth*[1], Brandon Haworth*[3], Muhammad Usman*[3], Davide Schaumann[2], Mahyar Khayatkhoei[2], Mubbasir Kapadia[2] and Petros Faloutsos[3]
[1]University Of British Columbia, [2]Rutgers University, [3]York University

1 Appendix

This section contains additional details related to efficiency considerations such as metrics parallelization, optimization framework and additional results.

1.1 Metrics Parallelization

The aforementioned metrics can be computationally expensive. The construction of the visibility graph $G_V(V, E)$ is $O(K \cdot N^2)$ where $N = |V|$ and K is the total number of obstacles in an environment. Furthermore, constructing the trees needed to calculate depth and entropy is of order $O(N \cdot b^d)$ where b is the maximum branching factor of the G_V and d is the maximum of all the minimum depths of the Trémaux trees constructed at different vertices. This process is $\Omega(N^2)$ which means that, at best, it is as complex as constructing the graph itself, although it is much more complex in practice. In order to mitigate this computational overhead, we off-load the construction of the visibility graph and the forest to the GPU.

For the purposes of parallelization, we consider that computing the metrics involves two main tasks: the construction of the visibility graph G_V for the given environment layout, and the computation of a set of N trees. Although we discuss each task separately, our implementation runs these computations concurrently, and not in isolation.

1.1.1 *Graph Construction*

We represent the strictly upper triangular part of the $|V| \times |V|$-dimensional symmetric adjacency matrix M_{adj} of the graph in row major fashion as a vector, V_{adj} of dimension that is equal to $0.5 \times (|V| - 1) \times |V|$. Each pair of (i, j) vertices in V where $i \leq j$ is assigned to a thread which calculates the straight-line between the vertices and checks whether the line intersects any obstacle. The load assignment is designed to exploit memory alignment and maximize GPU utilization.

1.1.2 *Tree Construction*

Consider the task of performing a Breadth First Search starting at a vertex $s \in V$ and branching until the whole visibility graph is traversed, i.e. all vertices are visited exactly once. We introduce three binary $|V|$-dimensional vectors: (a) frontier F^x holds the elements of V that must be expanded in the current level l, (b) children C^l holds the elements of V that must be expanded in the next level $l+1$, and (c) parents P holds the elements that have been already expanded. We also keep a $|V|$-dimensional integer vector D which stores the number of elements visited at each level.

A Naive Kernel. In a CUDA kernel, we assign each vertex, i in V to one thread, that is each thread i is responsible for one row of the adjacency matrix corresponding to the vertex i. The kernel runs, level after level, until a flag is set showing that all vertices have been visited. At each level l, each thread i first checks if its vertex i is adjacent to the j-th vertex of the graph, second if the j-th element is to be expanded ($F^l(j)$ is set), third if the j-th element has not been expanded ($P(j)$ is not set), and if so, the thread will set the i-th element to be expanded at the next level (set $C^l(i) = P(j) = 1$). After each level, the number of 1s in C^l is stored in $D(l)$, then the child vectors are copied into the frontier ($F^{l+1} = C^l$), and the child vector is reset (C^{l+1} is initialized to zero vector). Note that other information can be stored depending on the required metrics, but in this case the number of visited vertices at each level suffices.

Cut-Off Threads. In the naive kernel, each thread has to check exactly $|V|$ vertices of the frontier at each level, resulting in exactly $L \times |V|$ operations where L is the number of levels. However, each vertex in the graph only needs to be visited once. This fact can be exploited by cutting off threads that have already been visited from the start of each level, that is, stopping thread i whose assigned vertex has already been expanded ($P(i)$ is 1) from launching in the first place. This results in each thread having to check at most $|V|$ vertices of the frontier at each level, and in practice greatly reduces the running time.

Indexed Frontier. So far, each thread, if not cut off, has to check all $|V|$ elements of the frontier, even though many may be zero (not to be expanded) at many levels. However, each vertex in the graph can only be expanded once, that is, each element of the binary frontier can be 1 exactly once over all levels. Thus, the frontier is changed from a binary vector to an integer vector which stores the indices of the elements to be expanded. This indexed frontier is populated by an intermediate process that first sets all elements of frontier to 0, then starts filling it from the start with the indices of the positions of 1s in children, instead of just copying the children vector into the frontier at the end of each level. When a zero is encountered in the frontier (i.e. $F(j) = 0$), the kernel is terminated. This process essentially takes the burden of passing over the whole frontier from every single thread, to one single preprocessing thread. The result is that no thread will pass over the frontier more than once.

Forest Construction. The tree construction process must start at all vertices in the graph. Because one tree construction is completely independent of another, all the tree constructions may run concurrently. Therefore, the same kernel as before is used but a new dimension is introduced to all containers (this is essentially concatenating), and then we put each tree process on one row of the device grid. Thus, when the kernel runs at one level, all of the forest is expanded one level deeper on the device. This allows for having a very large pool of threads, and therefore maximizes the load sharing and consequently the GPU utilization.

Performance. Note that certain operations in our calculations (e.g. entropy calculations) are especially amenable to GPU parallelization. Moreover, the reported times include the initialization process for each granularity which is executed once per optimization; therefore the actual average times over objective calls would be considerably lower. In our current implementation, the spatial objective are computed concurrently on the GPU and a weighted sum of the spatial metrics may be used for efficiency. The performance analyses reported here encompass the entire spatial analysis pipeline averaged over 5 runs.

GPU Memory Complexity. The GPU memory required for the objective calculation on the GPU is of $O(N^2)$, more strictly it grows with $8N^2$, where N is the total number of vertices included in the graph. All example environments in Table 1 take up less than $20\ MB$ memory. Note that the provided memory complexity is for one unified run of optimization, a much larger environment can be processed in subsets (chunks) of vertices.

1.2 CMA vs Simulated Annealing + MCMC

The choice of optimization algorithm to use for this type of design problem is an important consideration. A recent review of building architecture related optimization frameworks highlights the numerous optimization techniques used in the area, and reasons why some are better than others for particular design problems [1]. Here we list the most relevant reasons for using CMA. Simulated annealing (SA) may need careful design of special parameter selection methods, like the ones used in [2]. SA is a poor choice given our desire for imposing design constraints. SA can handle noisy objective functions but only under certain conditions that can not be guaranteed for most building metrics. Also, genetic algorithms (GAs), like CMA, are often parallelizable, making the method more efficient. CMA should be better at escaping local-minima. Last, GAs have also been shown to show better early convergence, leading to quickly finding good local-minima that are often good enough for these types of design problems. CMA is a form of MCMC where the chain is the series of generated covariance distributions [3]. You can even formulate MCMC to use a variant of CMA for sampling to improve convergence [4] These samplers outperform many variants of MCMC [5]

1.3 Multi-Objective Optimization Methods

There exists several methods that can be used to perform multi-objective optimization [6]. Scalarized multi-objective optimization combines a vector of objectives with a vector of weights, however, finding a good vector of weights can be challenging, especially when the objectives are of largely different scales, as they are in our case. Pareto Front-based approaches produce a collection of parameter settings that are optimal trade-offs between the objectives [7]. However, they tend to be computationally expensive, and it is unclear how they would handle the diversity term.

1.3.1 Scalarized

Computes a weighted combination of the objectives, weighting all of the objective terms with respect to some relative weighting. This method is challenging to use for two reasons. One, determining the weights to use for a combination of objectives can be a daunting task. Also, the objectives themselves may not be linear, with some growing faster than others usually precluding the possibility of finding a single set of weights that works well when the environment changes. Second, If a relative weighting is used it helps to normalize the metrics in some way. The maximum value for the Degree metric can be found by removing all of the items from the simulation and calculating the degree. There is no simple calculation to find the diversity bound, however, an upper bound can be found via optimization. The diversity metric is very cheap to compute (relative to Degree, etc), an optimization for only diversity can be performed first, to find the upper bound on diversity. Both degree and diversity are non-linear functions, this is okay and could give desirable results when performing a scalarized optimization, but it would still be challenging to find objective weights [8].

1.3.2 Pareto Front Optimization

This method essentially finds a set of points (non-dominated points) that are optimal trade-offs between a set of objectives. The issue with using a Pareto Optimal Front method is that the computation of diversity between the members is non-trivial. Diversity is a measure of the distance between points in the Pareto front. It is not clear how to accomplish this without introducing a large number of parameters. Possibly, two different objectives could be chosen to optimize with respect to, but those objectives are only proxies for diversity and could be very similar producing results with minimal dissimilarity.

1.3.3 Hierarchical Optimization

With hierarchical optimization an ordering and objective specific thresholds are used, instead of only relative weights. The objectives are optimized in the order given. Each objective is optimized to find its optimum and from this a constraint is added to the optimization for the next objective. This constraint adds a penalty whenever the previous objective(s) value goes below the threshold value(s). This gives more control over the trade-offs between objectives. This method works well and converges quickly, as can be seen in Fig. 1. In this experiment we optimized *art-gallery B* in Fig. 6 with a diversity set of size 5. This optimization completed in a few seconds and converged before the optimization was terminated.

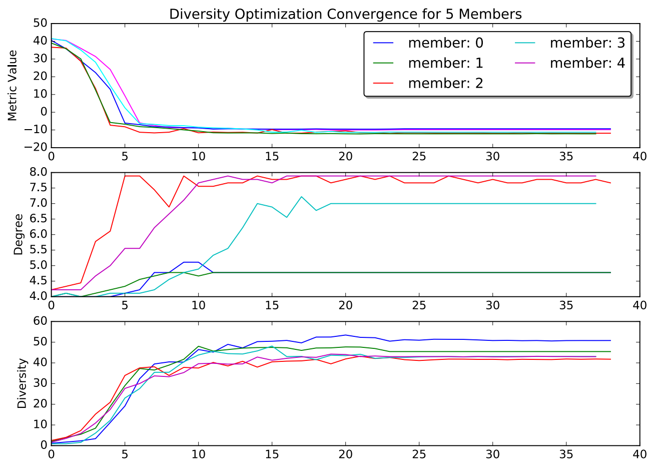


Fig. 1. Diversity optimization convergence

1.4 Critical Resolution

The grid resolution determines the number of vertices in the visibility graph, to identify the minimum resolution needed we perform a sensitivity analysis over the granularity. Each metric is computed over a range of grid resolutions, aggregated over multiple environment layouts. Here, resolution is represented as the number of cells per meter ratio, for example resolution 0.5 means that in each dimension one cell covers 2 square meters. The study results are illustrated in Fig. 2. These diagrams show that the metrics do not substantially change after a certain sampling frequency, suggesting that a critical value can be identified. The two jumps in the depth and entropy diagrams are caused by discovering new bottlenecks after a certain increase in resolution, which are discarded at higher sampling frequency. For the remaining experiments reported in this paper, we have used a sampling resolution of 0.5 cells/meter^2.

1.5 Additional Iterative User-In-The-Loop Results

We demonstrate an additional application of IDOME on a real-world environment, namely a portion of the NYC Penn Station subway. The user-in-the-loop approach affords an iterative design process, where a user may initially set up the problem by defining the movable elements, and the *Region of Query* and *Region of Reference*. Upon selecting a suitable revision to the layout from a set of diverse exploration candidates provided by the system, the user may modify the problem formulation. Fig. 3 illustrates results from three iterations. By adding additional parameters or changing the regions in an effort the user can resolve issues that may have been identified over the course of previous optimization rounds. In this example, the user iteratively includes new query regions for the stairwell and elevators to account for additional aspects of the layout. What appear as minor alterations to the wall configuration in the subway increase the objective from 6.3 to 11.68 leading to a design that significantly improves the pedestrian environment. (Additional information on this study is available in the accompanying video.)

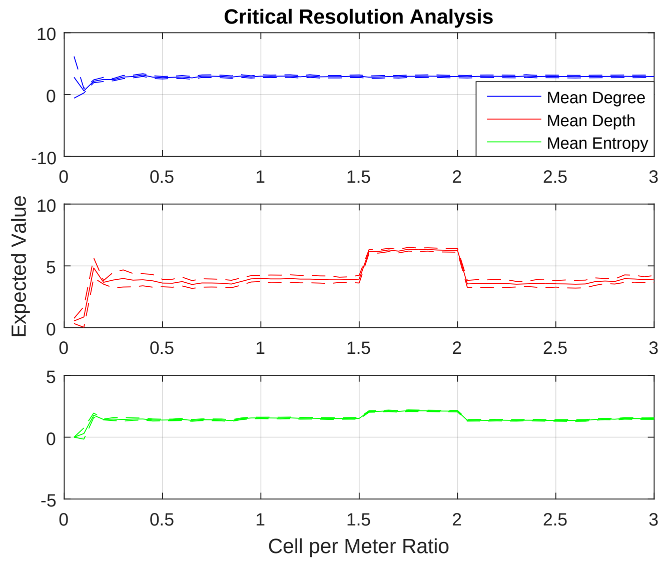


Fig. 2. Sensitivity of metric values to visibility graph resolution. The vertical axis is the metric average over 10 randomly sampled environments. The standard deviation is also provided for each metric.

1.6 Expert Usefulness Results

1.6.1 Preferences

To facilitate the understanding and perception of results of the art gallery study we render DOME designs in 3D and show a common viewpoint from which several interesting patterns can be seen. This is shown in Figure 4. Additionally, we solicit feedback from six expert architects via a design preference survey. The results show that all experts prefer IDOME results over the default environment design.

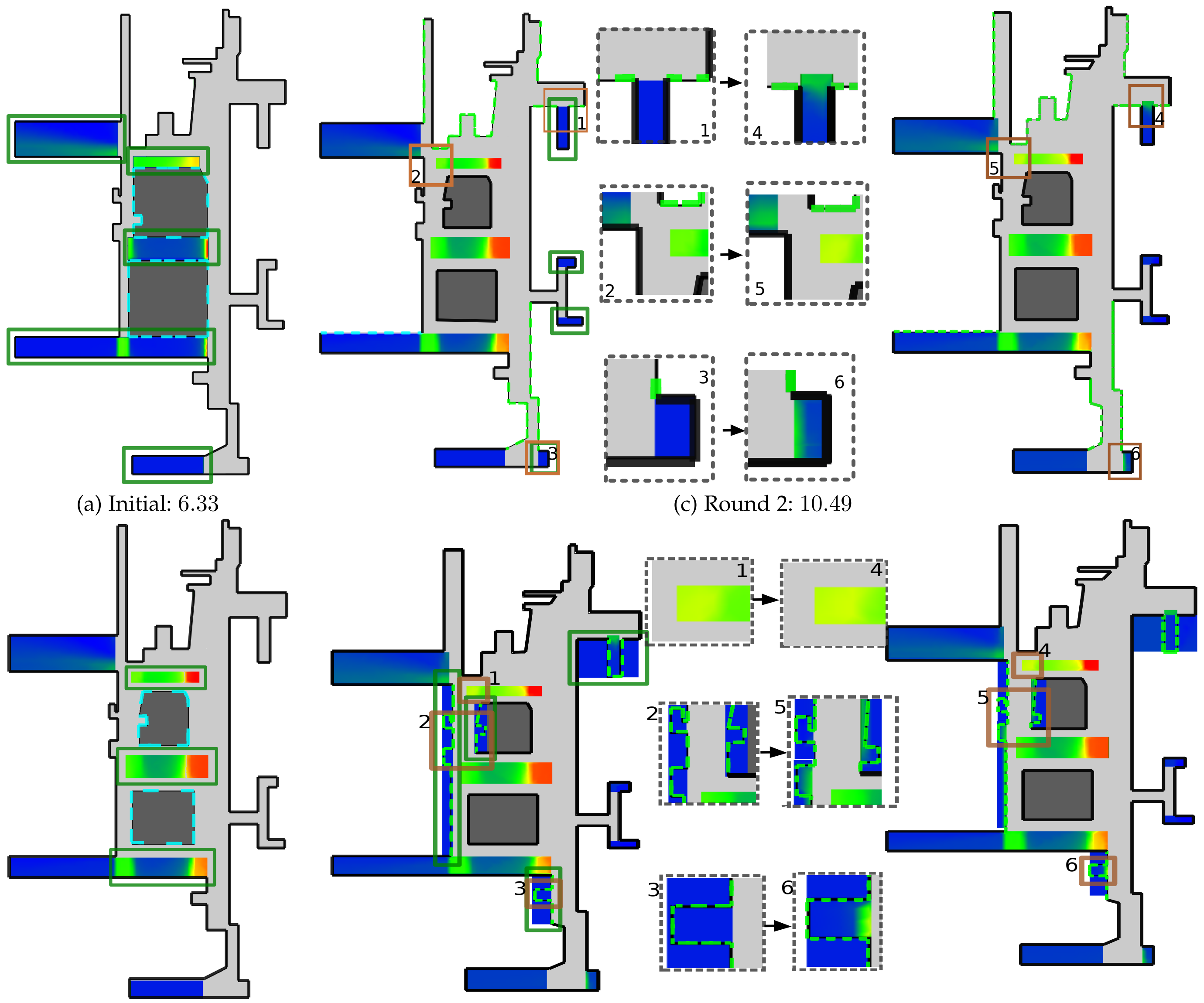
1.6.2 Usefulness

This study involved three architects from three different firms. The participants self-reported as male-identified with one in the range of $35 - 44$ and the others as $25 - 34$ years of age. All participants have a Master of Architecture degree from accredited universities. The answers to expertise questions relevant to both general expertise and IDOME-specific expertise can be found in Figure 5.

The expert usefulness survey was constructed to elicit the opinions of expert directly and indirectly about the IDOME approach. It is important to note that neutral results came largely from the prototype interface used in the study.

References

[1] A.-T. Nguyen, S. Reiter, and P. Rigo, "A review on simulation-based optimization methods applied to building performance analysis," *Applied Energy*, vol. 113, no. Supplement C, pp. 1043 – 1058, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0306261913007058
[2] T. Feng, L.-F. Yu, S.-K. Yeung, K. Yin, and K. Zhou, "Crowd-driven mid-scale layout design," *ACM Trans. Graph.*, vol. 35, no. 4, pp. 132:1–132:14, Jul. 2016. [Online]. Available: http://doi.acm.org/10.1145/2897824.2925894



(a) Initial: 6.33

(c) Round 2: 10.49

(b) Round 1: 7.21

(d) Round 3: 11.68

Fig. 3. Optimization of Penn Station, NYC. This figure illustrates how the framework can be used on a large complex environment of $\sim 10,000$ vertices. Additional *Region of Query* are incrementally added, to resolve issues in the layout that were identified during the previous design optimization rounds. The light grey area is the *Region of Reference*. The heat map areas are *Region of Query* with significant changes outlined in brown rectangles. The dashed cyan lines show the structure of interest that was optimized between each round. The green boxes highlight the new areas of interest that were considered during the optimization round. Round 1(a-b) regions are chosen to increase the accessibility and visibility of subway platform access. Round 2(c) regions are chosen to increase the accessibility and visibility of exits. Round 3(d) the placement of washrooms and elevators are improved by making them more viewable and accessible from additional areas in the environment. (Additional information on this study is available in the accompanying video.)

[3] O. Krause, D. R. Arbonès, and C. Igel, "Cma-es with optimal covariance update and storage complexity," in *Advances in Neural Information Processing Systems 29*, D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, Eds. Curran Associates, Inc., 2016, pp. 370–378. [Online]. Available: http://papers.nips.cc/paper/6457-cma-es-with-optimal-covariance-update-and-storage-complexity.pdf

[4] C. L. Mller and I. F. Sbalzarini, "Gaussian adaptation as a unifying framework for continuous black-box optimization and adaptive monte carlo sampling," in *IEEE Congress on Evolutionary Computation*, July 2010, pp. 1–8.

[5] E. Milgo, N. Ronoh, P. Waiganjo, and B. Manderick, "Adaptiveness of cma based samplers," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, ser. GECCO '17. New York, NY, USA: ACM, 2017, pp. 179–180. [Online]. Available: http://doi.acm.org/10.1145/3067695.3075611

[6] R. T. Marler and J. S. Arora, "Survey of multi-objective optimization methods for engineering," *Structural and Multidisciplinary Optimization*, vol. 26, no. 6, pp. 369–395, 2004. [Online]. Available: http://dx.doi.org/10.1007/s00158-003-0368-6

[7] T. Wagner, N. Beume, and B. Naujoks, "Pareto-, aggregation-, and indicator-based methods in many-objective optimization," in *Evolutionary multi-criterion optimization*. Springer, 2007, pp. 742–756.

[8] Y. Ding, S. Gregov, O. Grodzevich, I. Halevy, Z. Kavazovic, O. Romanko, T. Seeman, R. Shioda, and F. Youbissi, "Discussions on normalization and other topics in multiobjective optimization," in *Fields-MITACS, Fields Industrial Problem Solving Workshop*, 2006.

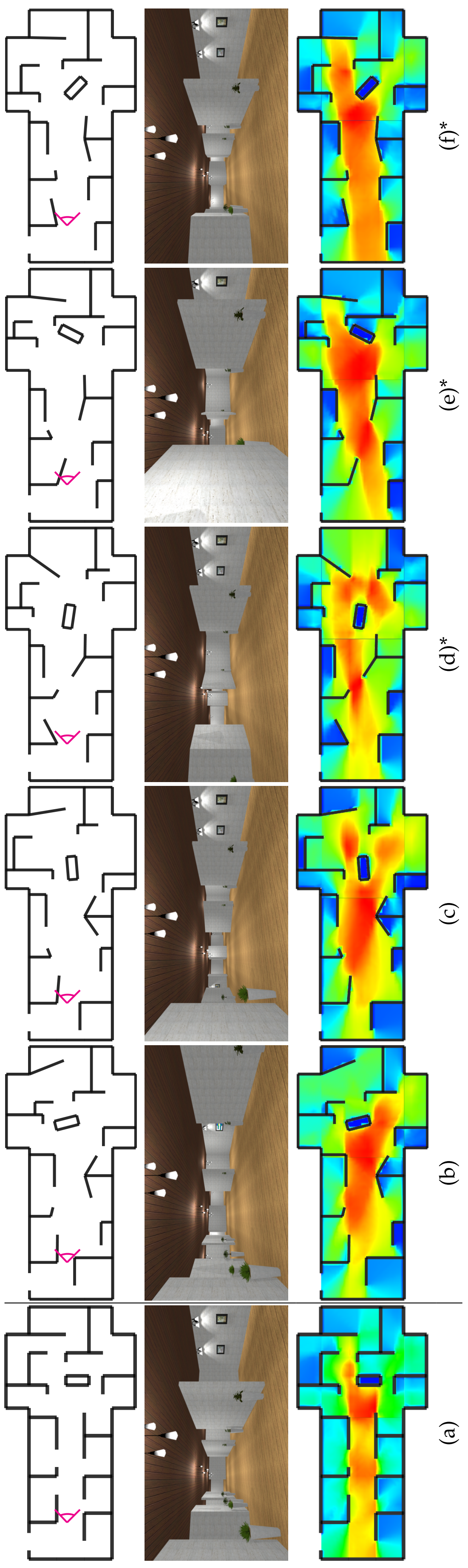


Fig. 4. Optimizing an art gallery and exemplifying the usefulness of having diverse near optimal candidates. The top row of figures shows the blueprint of the wall designs for the art gallery with a particular viewpoint shown in magenta. The middle row of figures shows the rendered environment from the viewpoint shown in the top row. The final row of figures shows the combined metric values as a heatmap over the entirety of each design. Column (a) is the original design of the art gallery. The columns (b) - (f) show the diversity members provided by the IDOME system for a particular parametrization of the environment. (b) is a member that opens up the floor space and the overall visibility down the corridor of the gallery. (c) is a member that balances the corridor visibility of (b) with the visibility of particular exhibits. (d) is a member that balances the visibility from (b) while reducing the number of path decisions further down the corridor and being particularly accessible. (e) is a member that mainly reduces path decisions while increasing gallery sizes. (f) is a member that balances the best of all designs being open, accessible, and easy to navigate. The (*)s identify the designs that six expert architects independently designated as preferred.

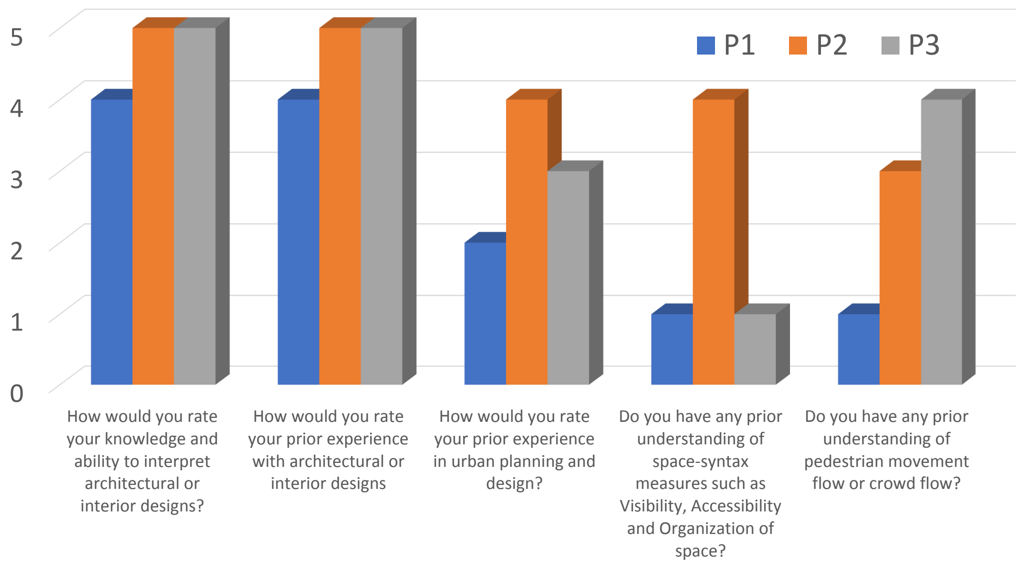


Fig. 5. Results for the expertise focused demographics questions for the expert usefulness participants (P1–P3).