Adaptive Exploration of Locally Satisfiable Proposals for Design Layout

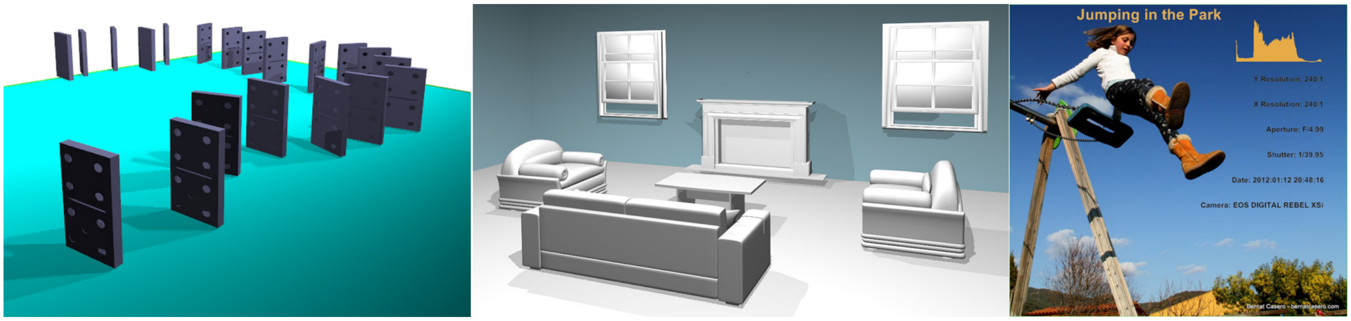


Figure 1: *Rule-based design layout algorithms can solve everything from laying out domino tiles, through arranging furniture in a room to superimposing metadata on a photograph following aesthetic principles. We formulate the problem as an iterative move-making algorithm and present a new approach to generating locally-satisfiable proposals for each object.*

Abstract

We present a new approach for rule-based design layout problems, examples of which include automated graphic design, furniture arrangement and synthesizing virtual worlds. We formulate this as a weighted constraint-satisfaction problem which is high-dimensional and requires non-convex optimization. We use a hyper-graph to represent a set of objects and the design rules applied to them. Although inference in graphs with higher order edges is computationally expensive, we present a hybrid solution-space exploration algorithm that adaptively represents higher order relations compactly. We exploit the fact that for many types of rules, satisfiable assignments can be found efficiently. In other words, these rules are *locally* satisfiable. Therefore we can sample from the known partial probability distribution function for each rule. Experimental results demonstrate that this sampling technique reduces the number of samples required by other algorithms by orders of magnitude. We demonstrate usage via different layout examples such as superimposing textual and visual elements on photographs. Our adaptive search algorithm is applicable to other optimization problems and generalizes many previously proposed local search based algorithms like graph cuts and iterated conditional modes.

CR Categories: F.4.1 [Mathematical Logic]: Logic and Constraint Programming— [G.3]: Probability and Statistics—Markov Processes;

Keywords: weighted constraint optimization, layout synthesis, graphic design

1 Introduction

Computer-assisted (or automated) design, layout synthesis and object arrangement problems consist of assigning values to object properties, guided by a set of rules. These rules are domain specific and may for example be geometric, examining the spatial relations between objects; color-based, matching the object's appearance to the palette of the design or enhancing the contrast of specific objects; or semantic, taking into account the functionality of the object. Regardless of their origin, they are represented as cost functions that measure the quality of any configuration of one or more objects. This paper focuses on instances of these problems in which a single 'good' solution is required, with low computational and time costs, mostly targeted at mobile devices.

Graphic design requires an eye for aesthetics and is more art than science. In most cases, creating a design combining textual and visual elements is a job better left to professionals. However, there are applications in which there is need real time layout of textual and graphical elements, based on predetermined rules. In these cases it is not possible to design each and every instance. For example, when browsing through a photo collection, it is useful to be able to view the image meta-data (title, exposure information, histogram), both on the camera and on various other devices. However, simply superimposing elements on a photo can obstruct faces and other salient regions and produce an unattractive layout. We aim to define rules that reference an image's saliency map, color palette and major edges, and lay out the data by observing symmetry, saliency, photographic principles and other quantifiable measures.

We formulate design layout as a weighted constraint-satisfaction problem. Our formulation incorporates both low-order and high-order interactions between design elements. Low-order rules are defined over one or two design elements. For example a rule that states that two particular objects should be close to each other. On the other hand, high-order interactions are defined over large number of design elements. For example a rule specifying that a group of objects needs to be collinear.

We represent a group of design rules operating on a set of objects using a hyper-graph. In this graph, each node represents an object and each edge is a rule, connected to all the objects it references. Inference in graphs with higher order edges is computationally expensive. We propose an adaptive solution space exploration algorithm that iteratively explores the solution space and in doing so is able to represent higher order relations compactly. We apply our adaptive search algorithm to design layout problems, however it is applicable to other optimization problems and generalizes many previously proposed local search based algorithms like Graph cuts [Boykov et al. 2001; Szeliski et al. 2006; Woodford et al. 2008; Gould et al. 2009; Lempitsky et al. 2010] and Iterated conditional modes [Szeliski et al. 2006; Jung et al. 2009].

The key idea that drives our exploration algorithm is an observation that for many types of rules, satisfiable assignments can be found efficiently. In other words, these rules are *locally* (individually) satisfiable *i.e.* we generate proposals for objects that locally satisfy individual rules with no need for blind sampling. Our method generates, over several iterations, locally satisfiable and random proposals for each object (node), until it converges to a solution that cannot be improved. Experimental results demonstrate that our *locally consistent* sampling technique is very efficient and requires substantially fewer number of samples compared to other algorithms. We demonstrate our approach on 2D layout problems involving superimposing textual and visual elements on photographs as well as comparing to previous results in furniture arrangement.

2 Related Work

Constraint Satisfaction Design and layout synthesis consist of rules referencing a set of objects. An assignment to each object can be measured by how well the rules are met, whether they are satisfied or violated. In essence these are constraint satisfaction problems (CSP) [Mackworth 1977], fundamental in Artificial Intelligence and Operations Research. A variant of the problem, weighted CSP defines a cost function assigned to each constraint, and the objective is to minimize the overall cost. A large majority of CSP algorithms [Kumar 1992] use a search paradigm over a limited set of possible object assignments. These approaches are relatively rigid, and do not offer interactive performance.

MAP Inference In computer vision, many tasks such as segmentation of an image can be formulated as image labeling problems where each variable (pixel) needs to be assigned the label which leads to the most probable (or lowest cost/energy) joint labeling of the image. The models for these problems are usually specified as factor-graphs in which the factor nodes represent the energy potential functions that operate on the variables [Kschischang et al. 2001]. In most vision models, the energy function is composed of unary and binary terms and the interactions between objects are generally limited to variables in a 4 or 8 neighborhood grid.

The sparse grid-like structure of the object interactions and the limited number of labels allows for fast solution of image labeling problems using techniques such as graph-cuts [Boykov et al. 2001; Gould et al. 2009; Lempitsky et al. 2010; Szeliski et al. 2006; Woodford et al. 2008], belief-propagation [Pearl 1982], and tree message-passing [Wainwright et al. 2005; Kolmogorov 2006]. In our case rules can be defined over multiple variables, and create complex factor graphs which these approaches do not handle well. Further, each object typically has a large space of possible configurations, which increases the complexity in multi-object interactions. Furthermore, in all but the simplest scenarios the factor graph contains cycles that makes the problem NP-hard even if the label space for each object is small.

Layout Synthesis Ultraviolet [Borning and Freeman-Benson 1998] uses a constraint satisfaction algorithm framework for interactive graphics. The constraints for user interface layout usually form a non-cyclic graph, are hierarchical in nature and container based and are therefore less complex. In [Yu et al. 2011; Merrell et al. 2011] a set of rules and spatial relationships for optimal furniture positioning are established from examples and expert-based design guidelines. These rules are then enforced as constraints to generate furniture layout in a new room. [Yu et al. 2011] employed a simulated annealing method which is effective but takes several minutes, while [Merrell et al. 2011] sample a density function using the Metropolis-Hastings algorithm implemented on a GPU. They evaluate a large number of assignments and achieve interactive rates (requiring a strong GPU). Both papers work with a small number of objects in relatively small rooms and in static scenarios. In [Yeh et al. 2012], the motivation is to populate a scene with a variable number of objects (open universe). They present a probabilistic inference algorithm extending simulated annealing with local steps.

3 Layout Specification

We define a *design* as a set of rules that are created by an artist or designer. The design is used to find a *layout*, an assignment of values to a set of objects, that satisfy the rules of the design. Given an environment in which we need to lay out the objects, we call the space of all possible assignments the *layout solution space*. We define a cost function for the design

$$c(s) := \sum_i r_i(\hat{s}) \tag{1}$$

where $r_i : (O_i \subseteq O) \to \mathbb{R}$ is a cost function (rule) operating on a subset of the objects, $s \in S$ is a specific solution, and $\hat{s}$ is a slice of the solution containing only the objects in O_i. Typically each rule applies only to a small subset of the objects.

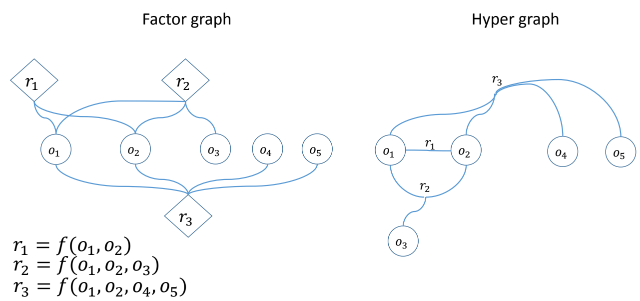


Figure 2: *A design (in this case consisting of three rules over five objects) can be represented as a Factor Graph, a bipartite representation connecting object nodes to factor nodes. Alternatively it can be represented as a hyper-graph in which each node is an object, and an edge represents common rules between connected nodes.*

A user specifies a design using declarative programming. First objects are defined, each identified by a unique name and belonging to one of several predefined classes. An object's class defines its properties which may be initialized to a specific value, and may be fixed (not allowed to change in the optimization process). For example an object of class *Title* might have properties such as position, rotation, font, font size, and color.

Rules are written using simple algebraic notation and a library of predefined routines, either as cost functions or as Boolean conditions (in which case we automatically assign a cost function). A rule can reference any of the objects defined (and their properties), as well as the environment. For example when arranging objects in a room the designer might reference wall and floor positions, in a 2D poster design the designer might reference the color palette of the background image. The number of objects included in a rule classify it as unary (one object), binary (two objects), ternary (three objects) or multiple.

3.1 Graph Representation

A common graph representation for MAP problems is the Factor Graph [Yeh et al. 2012] which has two node groups: object nodes and factor (rule) nodes. Edges connect factors to the objects they reference (Figure 2 left). A factor representing a unary rule will have one edge, a binary rule will have two edges and so on. We represent a design as a graph $G = (\mathcal{V}, \mathcal{E})$. Each object o has an associated node v_o whose cost function is $\phi(v_o) = \sum\{r | r : \{o\} \to \mathbb{R}\}$

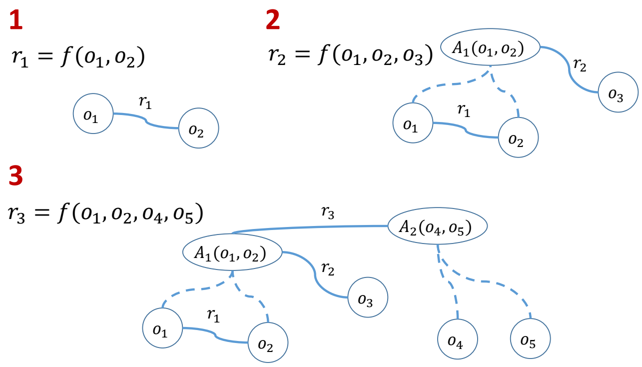


Figure 3: *We construct a graph for a design incrementally. Rules which reference more than two objects are transformed into pairwise interactions via auxiliary nodes: r_1 is a binary rule, r_2 is a ternary rule triggering the creation of A_1 which represents a pair of values (for o_1 and o_2). r_3 involves four variables, in which case we require two auxiliary nodes (we reuse previously created A_1).*

(sum of all unary rules on object o). We connect an edge e between v_{o_1} and v_{o_2} if there exists at least one rule associated with these objects. Its cost function is $\psi_e = \sum\{r | r : \{o_1, o_2\} \to \mathbb{R}\}$. Given an assignment to all of the objects, the summed cost over the nodes and edges of the graph is equal to the design cost (equation 1). Note that a rule may refer to more than two objects, and therefore an edge can connect more than two nodes, creating a hyper-graph (Figure 2 right).

4 Finding an Optimal Layout

An optimal layout is the global minimum in the scalar field defined by the design cost function. Finding the optimal layout or even a good one is difficult: Rule cost functions may be non-convex, rules might be unsatisfiable, for example if they conflict with the environment or with themselves, therefore we cannot know the lower bound on the cost and it is difficult to specify a stopping criteria. And finally, the high-dimensional nature of the space and the assumed sparsity of feasible solutions reduce the effectiveness of stochastic sampling.

Similar to [Merrell et al. 2011; Yu et al. 2011] we focus on a discretized version of the solution space. In some instances the application domain necessitates this (fixed resolution image backgrounds) while in other instances there is a specific resolution requirement which we need to meet. Given N objects in the design and k possible assignments per object, the size of the solution space k^N makes performing an exhaustive search prohibitively expensive. Previous methods have attempted to sample from the underlying probability distribution function, using Metropolis-Hastings [Hastings 1970] algorithm coupled with concepts from simulated annealing. These methods still require a prohibitively large number of samples (and of course evaluations of the cost function), therefore requiring a long run time or reliance on massively parallel GPU implementations [Merrell et al. 2011]. In many applications performance is an issue, and in some platforms such as mobile devices, computing is costly. Our approach therefore focused on reducing the number of evaluations required to find a feasible solution.

4.1 Transforming High-order rules into Pairwise Interactions

To simplify the graphical representation of the design, we transform hyper-edges into pairwise graph interactions by introducing auxiliary nodes. We divide the set of objects associated with a hyper-edge e into two groups A_1 and A_2. For each group consisting of more than one object we add an auxiliary node (otherwise we use the original object node). An assignment to auxiliary node A_i is an assignment to all variables associated with this node. The cost function of an auxiliary node is $\phi(A_i) = 0$ (no unary cost). We connect the two nodes with an edge $\hat{e}$ such that $\psi_{\hat{e}} = \psi_e$. We connect auxiliary nodes with their associated object nodes. The cost function for these edges $\psi(\{o, A_i\})$ is 0 if the assignment to object o matches the assignment to A_i and arbitrarily high otherwise. The addition of the auxiliary variables ensures that there are only binary interactions between nodes. Formally, this corresponds to a cost function:

$$E(x) = \sum_{i \in \mathcal{V}} \phi_i(x_i) + \sum_{ij \in \mathcal{E}} \psi_{ij}(x_i, x_j) \tag{2}$$

where $\mathcal{V}$ and $\mathcal{E}$ represent the set of nodes and the set of edges between these nodes respectively, x_i represents the label taken by a particular node, and ϕ_i and ψ_{ij} are functions that encode unary and binary costs. In figure 3 we demonstrate how the introduction of r_2 drives the creation of auxiliary node A_1, while the cost function is associated with the edge (A_1, o_3). Adding r_3 reuses A_1 while adding an additional node A_2 and connecting them.

4.2 Adaptive Layout Space Exploration

A simple method to find a low-cost solution under the function defined in equation 2 is to explore the solution space by local search *i.e.* start from an initial solution and proceed by making a series of changes which lead to solutions having lower energy. At each step, this *move-making* algorithm explores the neighboring solutions and chooses the move which leads to the solution having the lowest energy. The algorithm is said to converge when no lower energy solution can be found. An example of this approach is the Iterated Conditional Modes (ICM) algorithm that at each iteration optimizes the value of a single variable keeping all other variables fixed. However, this approach is highly inefficient due to the large label space of each variable. Instead we could perform a random walk algorithm, in each iteration we select a new value for one of the objects and evaluate the cost function. If the cost improves we save the new configuration:

Algorithm 1 Random Walk

```
minSolution ← RandomAssignment()
currentSolution ← minSolution
minCost ← Evaluate(minSolution)
for i ← 1, niters do
    for all O ∈ Objects do
        p_O ← nextProposal
        currentSolution ← p_0
        cost ← Evaluate(currentSolution)
        if cost < minCost then
            minSolution ← currentSolution
            minCost ← cost
        end if
    end for
end for
```

Generating proposals for this algorithm is key to its performance.

The most straight-forward approach is to sample uniformly over the object properties. Another approach is to start with uniform sampling (large steps) and over time reduce step size, sampling normally around the previous object value which we term *multi-resolution random walk*. Other heuristics exist and have been tried before, such as swapping between objects.

4.3 Exponential-sized Search Neighborhoods

Using a bigger move space has a higher chance of reaching a good solution. This observation has been formalized by [Jung et al. 2009] who give bounds on the error of a particular move making algorithm as the size of the search space increases. [Boykov et al. 2001] showed that for many classes of energy functions, graph cuts allow the computation of the optimal move in a move space whose size is exponential in the number of variables in the original function minimization problem. These move making algorithms have been used to find solutions which are strong local minima of the energy (as shown in [Boykov et al. 2001; Komodakis and Tziritas 2005; Kohli et al. 2007; Szeliski et al. 2006; Veksler 2007]).

While the traditional move making methods only dealt with variables with small label sets, their use has recently been extended to minimizing functions defined over large or continuous labels spaces [Lempitsky et al. 2010; Woodford et al. 2008; Gould et al. 2009]. An example of this work is the Fusion move method that in principle allows for the minimization of functions defined over continuous variables. Given a set of variables, the fusion-move algorithm works by proposing a labeling for all variables. It chooses for each variable whether to retain its previous label or take the new proposed label. Our method generalizes the above algorithms as, in each iteration, instead of proposing a single new label, it proposes multiple new proposals for each variable. Unlike [Veksler 2007], a large majority of these proposals are arbitrary values in the label space of each variable that are *locally satisfiable* (see section 4.4). However, the bigger difference is the fact that our method adaptively selects the number of variables included in the move. In this way it can smoothly explore the whole spectrum of choices between iterated conditional modes on one end, and the full multi-proposal fusion move, that involves changing the label of all variables.

Solving a single iteration We formulate the problem of jointly selecting the best proposals for all variables that satisfy the most rules as a discrete optimization problem. More formally, let $P_i = \{p_i^1, p_i^2, ..., p_i^k\}$ be a set of k proposal configurations for variable x_i. We introduce indicator variables $t_i^l, \forall i \in \mathcal{V}, \forall l \in \{1...k\}$ where $t_i^l = 1$ indicates that variable x_i takes the properties in proposal l. Similarly, we introduce binary indicator variables $t^{lr}, \forall ij \in \mathcal{E}, \forall l, r \in \{1...k\}$ where $t_{ij}^{lr} = 1$ indicates that variables x_i and x_j take the position proposed in proposal l and r respectively. Given the above notation, the best assignment can be computed by solving the following optimization problem:

$$
\begin{aligned}
\min \sum_{i\in\mathcal{V}} \sum_{l} t_i^l \phi_i(p_i^l) + \sum_{ij\in\mathcal{E}} \sum_{l,r} t_{ij}^{lr} \psi_{ij}(p_i^l, p_j^r) \\
s.t. \quad \forall i, \qquad \sum_{l} t_i^l = 1 \\
\forall i, j, l, \quad \sum_{r} t_{ij}^{lr} = t_i^l \\
\forall i, j, l, r \quad t_i^l, t_{ij}^{lr} \in \{0, 1\}
\end{aligned}
\tag{3}
$$

The above optimization problem in itself is NP-hard to solve in general. Instead, we solve its LP-relaxation and round the fractional solution. For this purpose, we could use general purpose linear programming solvers. However, we used an implementation of the sequential tree re-weighted message passing algorithm (TRW-S) [Wainwright et al. 2005; Kolmogorov 2006] that tries to efficiently solve the linear program by exploiting the sparse nature of the interactions between variables. TRW-S guarantees a non-decreasing lower bound on the energy, however it makes no assurances regarding the solution (See [Szeliski et al. 2006] for detailed comparisons).

Algorithm 2 Large Moves

```
procedure LARGEMOVES(O, R)                          ▷ Objects, Rules
    G ← ConstructGraph(O, R)
    minSolution ← RandomAssignment(O)
    minCost ← Evaluate(minSolution)
    for i ← 1, niters do
        A ← ObjectsToOptimize(G, currentSolution)
        for all o_i ∈ A do
            P_i ← ProposeCandidates()
        end for
        for all r ∈ R do
            UpdateGraphCosts(G, Evaluate(r, {P_1, P_2, ...}))
        end for
        currentSolution ← TRWS(G)
        cost ← Evaluate(currentSolution)
        if cost < minCost then
            minSolution ← currentSolution
            minCost ← cost
        end if
    end for
end procedure
```

Therefore our revised algorithm works as follows (see algorithm 2): Given a design we construct a graph as described in subsection 4.1. **In each iteration** we generate a set of candidates for all objects to be optimized. Candidates are sampled randomly or through locally-satisfiable proposals as described in subsection 4.4. We evaluate the cost of each rule, for each tuple of values associated with it. Therefore a unary rule is evaluated k times (k being the number of candidates), a binary rule k^2 times (if each of its referenced objects has k candidates) and so on. These costs are transferred to the graph nodes and edges as described above. Note that for complex rules, this creates a challenging number of evaluations, which can go up to k^n (where n is the number of objects in the design). We found that by limiting the set of candidates for auxiliary nodes to O(k) tuple values did not detract from the efficiency of the algorithm, and kept our complexity at $O(nk^2)$ overall. We then attempt to find an improved assignment for our objects, based on the populated graph. In each iteration we use either Random-Walk or TRW-S dependent on the number of objects to optimize. We repeat this procedure, carrying forward the current best assignment, until a maximum number of iterations or evaluations is reached, or the current solution is of acceptable cost.

4.4 Locally Satisfiable Proposals

The space of possible placements (style, color etc.) of a design element (object) is very large and it may take a large number of proposals to obtain a good assignment for the object [Ishikawa 2009]. We overcome this problem by guiding the mechanism through which new proposals are generated. For many types of rules, assignments that satisfy these rules can be found efficiently. In other words, these rules are *locally* satisfiable. In simple terms, given an assignment to some of the objects referenced by r we can generate good proposals for the rest, without resorting to blind sampling in the

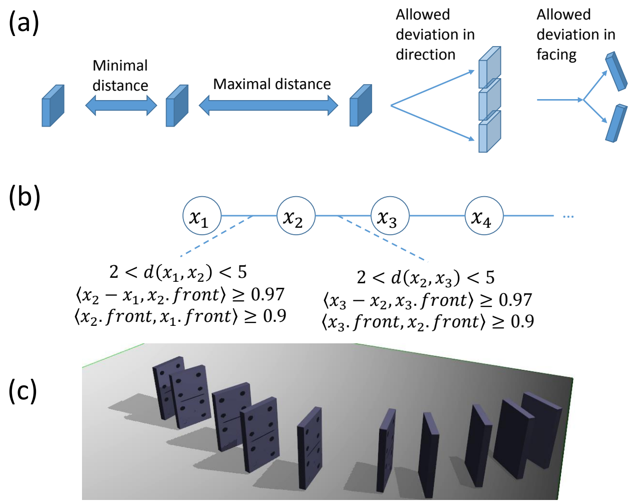


Figure 4: *A set of domino tiles laid out on a curve (a) Each domino tile must be within a set distance from its two neighbors, face in roughly the same direction, and remain roughly in line (b) Resulting graph is a chain where each each edge represents a compound rule between neighboring tiles (c) A result for ten tiles.*

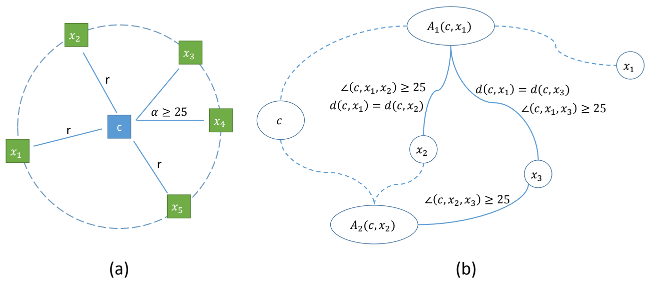


Figure 5: *A set of objects arranged in a circle around a central object (top right) The distance from each object to c is equal, the angle between each pair of objects x_i, x_j and c is at least 25 degrees (b) A partially labeled graph which shows auxiliary nodes created to represent ternary constraints. The dashed edges are auxiliary constraints, propagating the selected value of an object to its auxiliary nodes. This graph has high connectivity.*

layout solution space. Our approach could be seen as performing Gibbs sampling [Casella and George 1992], taking advantage of a known partial probability function, to sample from the whole solution space. For example a geometric constraint $dist(a, b) < 4$ is locally satisfiable as given a we generate proposals for b within the circle centered around a with radius 4. A color constraint $complementary(a, b)$ is locally satisfiable as given color a, b is easy to calculate. A unary constraint $saliency(a, bg) < 0.1$ attempting to place a 2D element a on a background image bg can generate proposals for a based on a pre-calculated saliency map of bg. When a designer declares rules in our declarative language, he can define a rule as locally satisfiable. This "inverse" function generates proposals for the rule referenced objects, given one or more object assignments. Rules which have not been declared explicitly can still be emulated as locally satisfiable by randomly sampling a large number of value tuples for the rule referenced objects, and picking the best tuples (which comes with a computational cost).

A *locally satisfiable proposal* (LSP) is a candidate for object o which was proposed by a locally satisfiable rule r. We generate

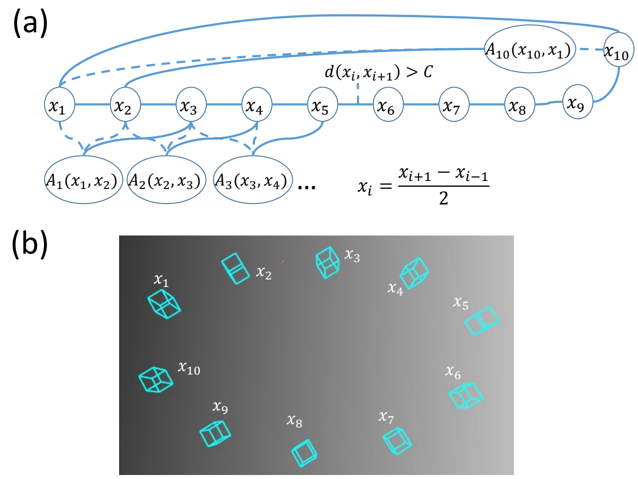


Figure 6: *(a) Ten objects are arranged such that each object attempts to be near the average of its two neighbors, and yet maintain a minimal distance from both. (b) The solution to this design is a least-squares one, satisfying none of the constraints fully.*

LSP using two strategies:

- Local: Given a current assignment to all objects, and a set of *active* objects to be optimized in this iteration, we iterate over all rules referencing these objects. For each rule r and for each object r references, we fix that object value and produce one or more LSP for the other objects.
- Greedy: Given the hyper-graph structure of the design, we apply a BFS starting from a randomly selected node. As we discover new nodes, we generate LSP for them, based on the nodes already visited, and the edges by which we discover these nodes. Repeating this algorithm creates a series of greedy assignments.

5 Experimental Evaluation

We measure the performance and quality of a layout optimization algorithm by counting rule evaluations. For example calculating the cost of a specific layout for a design is $|r_i|$, the number of rules in the design. Previous approaches have counted the number of samples the algorithm performs for all objects in all iterations. However, this measure favors algorithm which perform an exhaustive search over limited combinations of values. Another measure consists of counting number of evaluations of complete layouts. However, this is not representative of belief-propagation algorithms (such as TRW-S) in which partial evaluations are combined together.

Designs differ in the type and number of rules they contain, and by how constrained the solution is. These differences are reflected in the underlying graph structure, and in our ability to create locally satisfiable proposals. We tested our move-making algorithm on a large number of different designs, of which we present a few archetypes here. For each design we present the results of several algorithm configurations, varying over the number of objects optimized in each step, size of candidate set and usage of locally-satisfiable proposals. The results demonstrate that the connectivity of the graph is a good indicator for tuning the move-making algorithm, and that locally-satisfiable proposals reduced the number of evaluations required in all scenarios. In our experiments the rules are geometric, and each objects in the design can be assigned position, rotation and scale in 2D. We apply the configurations described in table 1 on the following designs. In each design we ran each variant algorithm 30 times and took the median of the results. Our algorithm is currently a non-optimized single-threaded CPU implementation, its average run time is 2-3 seconds.

Domino - Thirty tiles arranged in a curve i.e. each tile is at a certain distance from the next, and facing in a similar direction. The design and graph structure are visualized in figure 4. The created graph is a chain structure which can be optimally solved by belief propagation algorithms such as TRW-S. Moreover, non-cyclic graphs appear to work extremely well with *greedy* LSP, which is reflected in the results (figure 7).

Circle - In order to test a highly connected graph with cycles, we created a design for nine tiles arranged in a circle (non-fixed radius) around a central tile. The minimal angle between any two tiles is at least 25^o. The design and resulting graph structure are shown in figure 5, while the experiment results are in figure 8. All rules in this design are ternary, and the rules enforcing a minimal angle between all objects create a graph with high connectivity. In this scenario evaluating proposals for multiple objects at once is disadvantageous. We found that TRW fails to find a solution that approximates the minimum. However, exhaustive evaluation of each candidate set is not realistic. Therefore we reduced the number of candidates as shown in the graphs, to 4 and 6 candidates, and saw improvement. Still, constraining the algorithm to one proposal for a single object in each iteration gives the best results for this scenario. Locally satisfiable proposals prove to be an asset in this scenario, achieving an x2 factor from 20000 evaluations and up until 120000 evaluations.

Laplacian Cycle - Finally, to challenge the candidate proposal process, we attempt to arrange ten tiles $t_1..t_{10}$ such that $t_i = (t_{i-1} + t_{i+1})/2$ and $d(t_i, t_{i+1}) > C$. Since the rules wrap around t_{10} the cost can never be 0 and the best possible solution is a least-squares oval structure. The design and resulting graph structure are in figure 6, while the experiment results are in figure 9. We applied several variants of TRW on this design, starting from 2 proposals for each object each iteration (one LSP, one random) and up to 10 candidates. Additionally we tried multi-resolution random walk with and without LSP. We found that a small number of candidates over a large number of iterations produced the best result, and that LSP were of benefit.

Algorithm	#Obj	#Cands
Random Walk	1	2
Multi-Res RW	1	2
LS MR RW	1	2
LS TRW (Greedy)	n	k
LS TRW (Local)	n	k

Table 1: *Algorithms applied in our experiments. #Obj is the number of objects optimized in each iteration, #Cands is the number of candidates assigned to each (optimizable) object in each iteration. The TRW variants have k candidates, 50% of which are locally satisfiable proposals (if applicable).*

5.1 Furniture Arrangement

We recreated the design rules described in [Merrell et al. 2011], rewriting them to be locally satisfiable. We then used our algorithm to find furniture layout in several room configurations (figure 10). Our approach produced comparable results in 50000 evaluations, compared to $5M - 10M$ evaluations (extrapolated from figure 7 in their paper). Note that while we produce a single feasible solution in each run, they attempt to produce a variety of solutions.

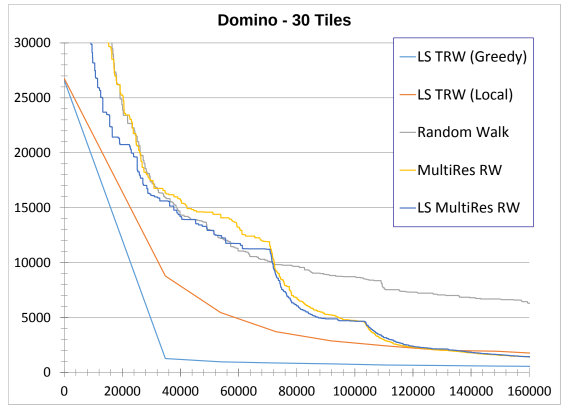


Figure 7: Domino*: The chain-like structure of the graph works optimally for belief propagation algorithms such as TRW. Combining TRW with greedy LSP converges on a solution quickly.*

5.2 Photo Overlay

When reviewing a large collection of photographs, it is often useful to see the corresponding meta-data (title, date, exposure information, geo-tags and user supplied tags) alongside the original image. In smaller form-factors, there is not enough screen real estate to display the image alongside the information. We present an application to overlay textual and visual information on an image, based on geometrical and aesthetic design rules.

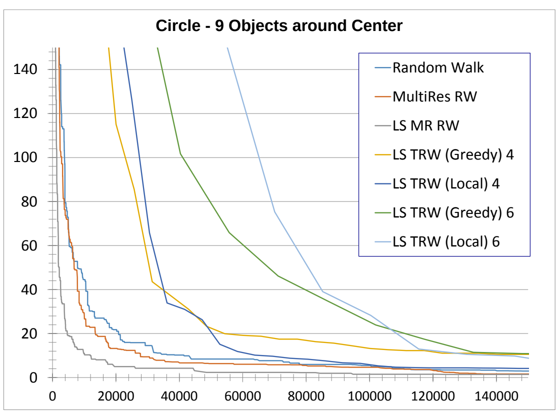
Given an image, we extract textual meta-data from its EXIF such as title, camera and lens model, aperture, shutter speed as well as calculate a luminance histogram. We then calculate a saliency map [Perazzi et al. 2012], run an edge detection filter and extract the color palette of the image [Morse et al. 2007] to which we add black and white (Figure 11. We define our design such that the superimposed elements are positioned on the non-salient regions in the image and their colors are taken from the image extended color palette. Additionally, the title is larger than the exposure information, and the exposure information elements are ordered vertically and attempt to align (left or right depending on their position within the image). The results of our design can be seen in figure 12 and figure 13.

6 Conclusions

We presented a new algorithm for rule-based design layout problems. Our approach unifies and expands on previously proposed local search based methods in the sense that it adaptively determines the search neighborhood according to the underlying graph structure. We introduced the concept of locally satisfiable proposals and demonstrated that their use dramatically reduces the number of evaluations required for finding a rule-consistent layout. In cases where LSP fails, our algorithm degrades to a random sampling approach.

While the efficacy of our method was demonstrated on a 2D layout problem, our solution generalizes to a wide range of layout prob-

Figure 12: *Applying our design rules to various images produces pleasing results automatically and in real time, allowing a user to bring up information about an image without interrupting the flow of images.*



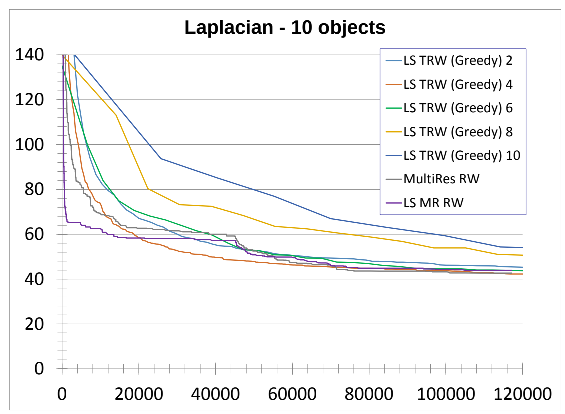


Figure 8: Circle*: The highly connected graph is a hindrance when using a large number of candidates. In this scenario increasing the number of optimizable objects in each iteration, and the size of the candidate set did not prove beneficial. The best performing variant was a multi-res random walk approach, in which the candidates were LSP.*

Figure 9: Laplacian Cycle*: The optimal solution in this design is a least-squares one, and creates a relatively complex graph. Still we find that LSP helps converge quickly, especially with a small set of candidates in each iteration (half of which are random).*

lems in two or higher dimensions. In future work we seek to apply our algorithm on layout synthesis in 2.5D (such as placing objects on a topographical map) and in 3D. Moreover, we are working towards a massively parallel implementation of our approach that we hope would be beneficial for the graphics community.

References

BORNING, A., AND FREEMAN-BENSON, B. 1998. Ultraviolet: A constraint satisfaction algorithm for interactive graphics. *Constraints 3*, 1, 9–32.

BOYKOV, Y., VEKSLER, O., AND ZABIH, R. 2001. Fast approximate energy minimization via graph cuts. *PAMI 2001*.

CASELLA, G., AND GEORGE, E. I. 1992. Explaining the gibbs sampler. *The American Statistician 46*, 3, 167–174.

GOULD, S., AMAT, F., AND KOLLER, D. 2009. Alphabet soup: A framework for approximate energy minimization. In *CVPR 2009*, 903–910.

HASTINGS, W. K. 1970. Monte carlo sampling methods using markov chains and their applications. *Biometrika 57*, 1, 97–109.

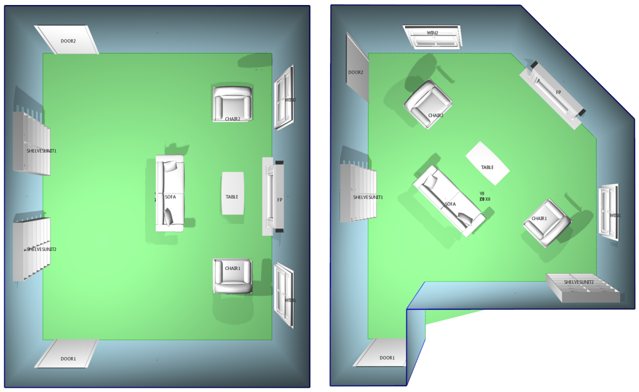


Figure 10: *We follow the design rules described in [Merrell et al. 2011] and apply our move-making algorithm to generate these furniture arrangements.*

Figure 11: *For each image we extract a saliency map [Perazzi et al. 2012] and a color palette [Morse et al. 2007].*

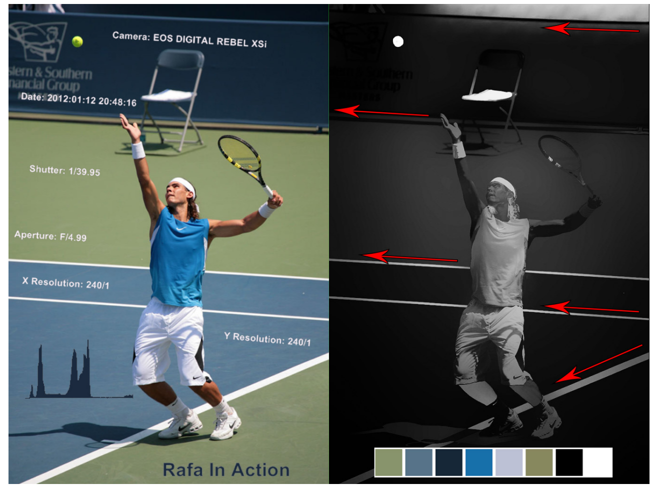


Figure 13: *In this example, in addition to the described design rules, we align the meta-data elements along the most prominent horizontal edges, extracted using edge-detection.*

ISHIKAWA, H. 2009. Higher-order gradient descent by fusion-move graph cut. In *ICCV 2009*, 568–574.

JUNG, K., KOHLI, P., AND SHAH, D. 2009. Local rules for global map: When do they work ? In *NIPS*, 871–879.

KOHLI, P., KUMAR, M. P., AND TORR, P. H. S. 2007. P3 & beyond: Solving energies with higher order cliques. In *CVPR*.

KOLMOGOROV, V. 2006. Convergent tree-reweighted message passing for energy minimization. *IEEE Trans. Pattern Anal. Mach. Intell. 28*, 10 (Oct.), 1568–1583.

KOMODAKIS, N., AND TZIRITAS, G. 2005. A new framework for approximate labeling via graph cuts. In *ICCV*.

KSCHISCHANG, F. R., FREY, B. J., AND LOELIGER, H.-A. 2001. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory 47*, 2, 498–519.

KUMAR, V. 1992. Algorithms for constraint satisfaction problems: A survey. *AI MAGAZINE 13*, 1, 32–44.

LEMPITSKY, V. S., ROTHER, C., ROTH, S., AND BLAKE, A. 2010. Fusion moves for markov random field optimization. *IEEE Trans. Pattern Anal. Mach. Intell. 32*, 8, 1392–1405.

MACKWORTH, A. K. 1977. Consistency in networks of relations. *Artificial Intelligence 8*, 1, 99 – 118.

MERRELL, P., SCHKUFZA, E., LI, Z., AGRAWALA, M., AND KOLTUN, V. 2011. Interactive furniture layout using interior design guidelines. In *SIGGRAPH 2011*.

MORSE, B., THORNTON, D., XIA, Q., AND UIBEL, J. 2007. Image-based color schemes. In *ICIP 2007*.

PEARL, J. 1982. Reverend bayes on inference engines: A distributed hierarchical approach. In *AAAI*, 133–136.

PERAZZI, F., KRÄHENBÜHL, P., PRITCH, Y., AND HORNUNG, A. 2012. Saliency filters: Contrast based filtering for salient region detection. In *CVPR 2012*.

SZELISKI, R., ZABIH, R., SCHARSTEIN, D., VEKSLER, O., KOLMOGOROV, V., AGARWALA, A., TAPPEN, M. F., AND ROTHER, C. 2006. A comparative study of energy minimization methods for markov random fields. In *ECCV 2006*.

VEKSLER, O. 2007. Graph cut based optimization for mrfs with truncated convex priors. In *CVPR 2007*.

WAINWRIGHT, M. J., JAAKKOLA, T., AND WILLSKY, A. S. 2005. Map estimation via agreement on trees: message-passing and linear programming. *IEEE Transactions on Information Theory 51*, 11, 3697–3717.

WOODFORD, O. J., TORR, P. H. S., REID, I. D., AND FITZGIBBON, A. W. 2008. Global stereo reconstruction under second order smoothness priors. In *CVPR*, IEEE Computer Society.

YEH, Y.-T., YANG, L., WATSON, M., GOODMAN, N. D., AND HANRAHAN, P. 2012. Synthesizing open worlds with constraints using locally annealed reversible jump mcmc. *ACM Trans. Graph. 31*, 4 (July), 56:1–56:11.

YU, L.-F., YEUNG, S.-K., TANG, C.-K., TERZOPOULOS, D., CHAN, T. F., AND OSHER, S. J. 2011. Make it home: automatic optimization of furniture arrangement. In *SIGGRAPH 2011*.