J. S. Gero, B. Tversky and T. Purcell (eds), 2001, *Visual and Spatial Reasoning in Design, II*
Key Centre of Design Computing and Cognition, University of Sydney, Australia, pp.

TOWARD AUTONOMOUS LAYOUT DESIGN

An AI Approach

SUSAN L. EPSTEIN
The City University of New York
USA

Abstract. Autonomous design poses challenging problems for the traditional AI problem-solving paradigm, but surmountable ones. Two systems for autonomous two-dimensional layout design are described, with particular attention to the ways in which they devise solutions. Human expertise inspired and informed both systems. Their pragmatic approaches offer important lessons for design problem solving

1. Introduction

Artificial intelligence (*AI*) problems are ambitious, ill-defined tasks for which there are, initially, no repertoire of efficient, effective algorithms. Examples include "play chess well" and "navigate through three-dimensional space." In many such areas, reasonably-expert AI systems now provide usable advice that supplements or even supplants human expertise. Design problems, with their myriad possibilities and their vague evaluation criteria, would appear to be prime AI targets. Nonetheless, the application of AI to design has almost always been restricted to *CAD*, computer-aided design. The thesis of this work is that computer programs could design autonomously, or participate as full members of a design team, rather than be relegated merely to CAD.

This paper explores how AI can move beyond visualization and calculation toward autonomous programs for two-dimensional layout design. The next section explains why, within the AI problem-solving paradigm, design is difficult. Subsequent sections consider two recent, quite different programs for two-dimensional spatial layout design, each of which goes beyond CAD to reusable techniques that support *CD* (*Computer Design*). The final sections describe future work and highlight some important lessons for those interested in moving toward autonomous design.

2. How Layout Design Challenges AI

Design problems differ from most traditional AI problems in a variety of ways. Although the following discussion focuses on two-dimensional spatial layout design, it may be extended to other design environments without loss of generality. This analysis considers how design impacts upon problem definition and solution approaches.

2.1 DESIGN AND PROBLEM DEFINITION

To begin, consider the *carousel problem*:

> Place a round carousel of radius 1 and a square ticket booth with side 1 in a recreation area that measures 4 by 6. The two structures may not overlap, and the carousel must be to the left of the ticket booth. The ticket booth should be convenient to the carousel, but not crowd it. Moreover, together the two should be well within the recreation area.

In AI, a problem's *representation* is the way a programmer specifies the relevant features of the problem. Using that representation, a description of the problem-solving world is called a *state*. One could represent a state in the carousel problem with a diagram that indicates the location of the carousel with a circle, and the location of the ticket booth with a square, both within a rectangle that portrays the recreation area. Figure 1(a) is such a representation for one possible carousel state. Alternatively, if a grid coordinate system were imposed on the recreation area, with its origin at the lower left corner of the rectangle, so that the carousel's center lay at *C*, and the ticket booth's center at *S*, one could represent a carousel state as <*C, S*>. Under this scheme, Figure 1(a) would be represented as <(1, 3),(3, 2)>, and Figure 1(b) as <(3, 2), 2, 2)>. If, in addition, ? is the symbol for "no location," Figure 1(c) would be represented as <?, (3, 2)>.

Typically, an AI problem has a single *goal test*, a way to confirm that a state is a desired state of the world (*goal state*). For example, a program that plays chess is expected only to win, and a program that navigates is expected only to produce a route. Design problems, however, ordinarily have multiple, vague goal tests (Goel and Pirolli, 1992). The state in Figure 1(a), for example, locates the two structures so that they do not overlap, and keeps the ticket booth to the left, but is not particularly responsive to the other problem specifications concerning distance, convenience, and crowding. The carousel problem requires the structures to relate to each other, and to the recreation area itself, in a variety of ways. Moreover, key words in the problem's definition, (e.g., "relatively," "convenient," "well within") are vague.

Design problems, as typified by the carousel problem, actually have two kinds of goal tests: required properties of a goal state (*constraints*) and a second set of non-required, vague but nevertheless important criteria on which a state may be judged (*principles*). Constraints are boolean tests or well-defined metrics with explicit limits. The carousel problem's strictures on "no overlap" and "to the left of the ticket booth" are examples of constraints. The other specifications in the carousel problem ("ticket booth convenient to the carousel, " "ticket booth does not crowd the carousel," and "ticket booth and carousel well within the recreation area") would all be classified here as principles — significant but imprecise features on which the quality of a goal state will be judged. In that case, Figures 1(a), 1(e) and 1(f) would be goal states, and Figure 1(b) would not.

As typified by the carousel problem, goal tests in design problems are often ill-defined computationally. The definition of distance between the carousel and the ticket booth is vague — it could mean the Manhattan or the Euclidean distance between the structures' centers, or the minimum distance between their perimeters. Nor is there an obvious definition for the overall distance between several objects and the perimeter of the area that contains them. The carousel problem is also vague about the relative importance of these requirements. Are they prioritized in some manner, or is there some weighting on them? In other words, would one prefer Figure 1(e) or Figure 1(f)?

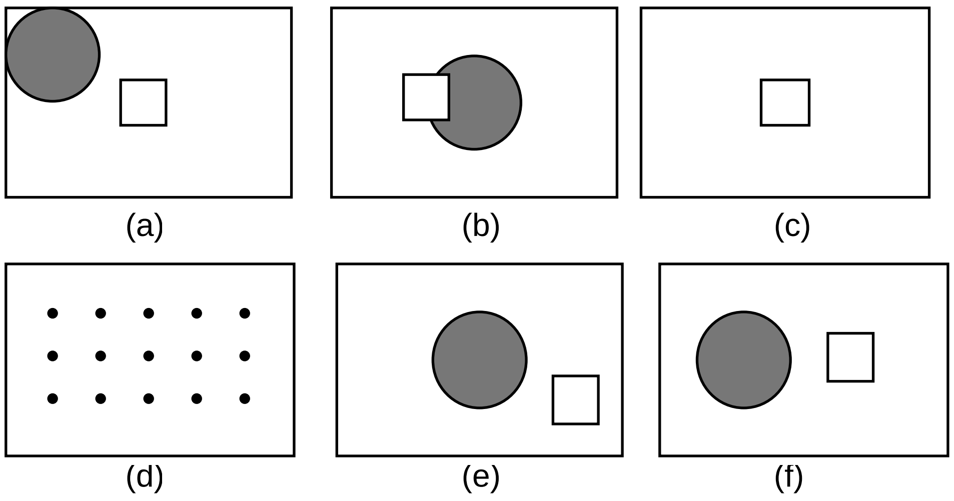


Figure 1. Some carousel problem states where the circle represents the carousel and the square represents the ticket booth. (d) shows the legal locations for the center of a carousel-problem structures.

Ordinarily, any goal state is acceptable to an AI problem solver. In design, however, goal states are judged by the degree to which they satisfy all of the problem's principles. That overall metric is referred to here as a state's *worth*. When a designer refers to one goal state as "better" than another, it is really a statement about the goal state's rating according to the principles of the design problem. The set of all possible states for a problem is called its *state space*. If locations in the carousel problem are restricted to integer coordinates within the rectangle, there are only 16 valid locations — ? and the 15 shown in Figure 1(d). Since either structure could be located at any of these, there are 16 16 = 256 possible states in the carousel problem's state space. Of these, 54 states are goal states, that is, they locate both structures without overlap, with the carousel to the left of the ticket booth. The worth of those 54, however, covers a broad range.

2.2 DESIGN AND PROBLEM SOLUTION

Typically, an AI problem solver transforms one state into another by taking an *action*. In the carousel problem, the only actions are to specify or change the location of the carousel, and to specify or change the location of the ticket booth. Figure 2 is a fragment of the carousel problem's state space; the lines define legal actions. (The space is actually a finite graph, portrayed for clarity here as an infinite tree.) For example, the line in Figure 2 between <?, (3, 2)> and <(1, 3), (3, 2)> means that a problem solver could change Figure 1(c) into Figure 1(a) by inserting the carousel at (3, 2). In AI, a problem is defined by one or more *initial state(s)* from which search will begin, a set of legal actions, and one or more goal states. For example, the initial state for the carousel problem might be the empty rectangle <?, ?>, and the legal actions would be to insert or shift a structure in the recreation area. In this way, AI views the solution of a problem as a *path* (a sequence of actions) through the state space, beginning at some initial state and ending at a goal state. Each choice of an action is a *decision.*

If a problem's state space is small enough, an AI problem solver can simply *search* for a solution, that is, use actions to move from one state to the next in the state space until it reaches a goal state. In the carousel problem, the program would simply begin at <?,?> and locate one structure at a time until it happened upon a goal state. There are no general search mechanisms guaranteed to find a solution in a reasonable amount of time.

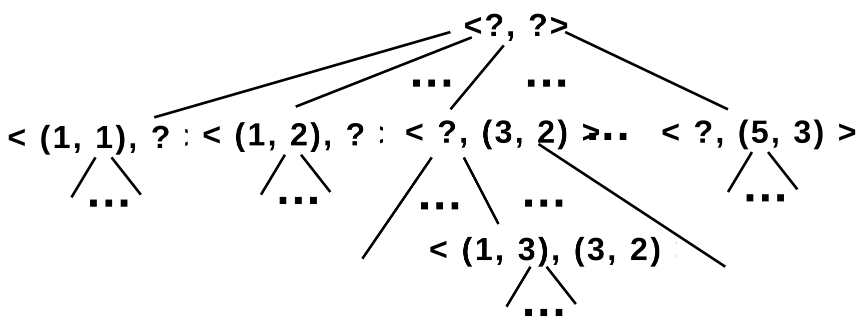


Figure 2. Part of the carousel problem state space.

Unless some rationale lies behind the choice of an action, search is termed *brute-force*. This lack of guidance becomes significant as the size of the search space increases, and as goal states become increasingly scarce. Consider, for example, the *fair problem*:

> Place a round carousel of radius 1, a Ferris wheel with circular footprint of radius 1, and seven different square vending booths, each with side 1, in a recreation area that measures 4 by 6. The structures may not overlap, and the booths should be convenient to the amusement rides, but not crowd them. Moreover, the rides should be well within the recreation area.

Although it is quite similar to the carousel problem, with the same integer-based grid the fair problem has $16^9 = 6.87 \quad 10^{10}$ possible problem states, only 80 of which are goal states. In this sense, the fair problem is quite typical of design problems: very few solutions in a very large space. Yet the fair problem involves only 9 structures, a reasonable number in, say, landscape design.

A typical AI search in the fair problem would soon arrive at a state where no new structure could be placed without overlap. In that case, the problem solver could *backtrack*, that is, remove the most recently positioned structure or structures. One would, of course, take precautions against repeatedly visiting the same state during search. Many well-known graph-search algorithms do that efficiently. Such search is guaranteed to find a solution when one exists. Even with backtracking, however, brute force search would still be unlikely to solve the fair problem.

One way to improve search is to have a problem solver select not an arbitrary action but one that appears to lead to a promising state. The quality of a state is subjective and problem-dependent, to be sure, but it is often quantifiable. An *evaluation function* assigns some numeric value to each state in the search space, a value that reflects the likelihood that further search from that state will lead to a goal state. In the fair problem, for example, a state where both rides have been placed and do not overlap is generally preferable to one where that is not the case. That is a *heuristic,* a good general rule of thumb, but not a foolproof predictor. Evaluation functions are often heuristic, because the knowledge to construct a perfect one is unavailable. Nonetheless, a good heuristic is likely to provide a measure of efficiency that makes state space search, and therefore problem solving, possible.

An expert problem solver solves a particular class of problems better and more quickly than the rest of us (D'Andrade, 1990; D'Andrade, 1991). AI has regularly demonstrated that expertise requires knowledge. In particular, the information embedded in a good heuristic evaluation function captures the kind of expertise that curtails search. The expert knowledge in the evaluation function suggested above was "get the large structures located early." One might presume then, that all a good AI system requires to replicate human expertise is a strong infusion of expert knowledge. The difficulty here is the *knowledge acquisition bottleneck,* the fact that extraction and representation of human expert knowledge for a machine is a costly, complex, and error-ridden process. Even the most cooperative human experts cannot always verbalize what is relevant, in what order of importance those things are relevant, or how newly-relevant knowledge is identified. This issue is particularly prevalent in design problems, where human designers would claim that their principles are not algorithmic, let alone quantifiable. What, for example, is "good spacing" or a "pleasing view"? And there always remains the possibility that even the power of knowledge may not pare a design problem down to a manageable size. For example, if a problem solver were to locate the rides in the fair problem first, that would still leave 16^7 states to consider.

Classical AI problems, such as chess or navigation, have a sequence of actions as a solution. In contrast, design problems do not require that decisions occur in a particular sequence — the sequence in which one located the structures in the carousel problem has no impact on whether or not the state itself satisfies a goal test. Along the solution path, classical AI problems abide by rules that are still in effect at the goal state, for example, "keep each chess piece within the confines of some square." Design problems are not restricted in that manner; one might consider a carousel state, for example, in which the structures overlap. Furthermore, most AI problems are best served if the search uses the same representation throughout the state space; here again design problems differ. In the carousel problem, for example, the metric imposed by the grid may ultimately be binding, but a human designer might initially choose to ignore it, concentrating on proximity instead.

Design problems, then, are large, non-sequential, and ill-defined. Nonetheless, a human expert presented with a set of constraints and principles does produce a variety of high-quality solutions for a particular two-dimensional layout design problem. Although the traditional AI problem-solving paradigm, informed search through a state space, seems ill-suited to design problems, there are constructive, pragmatic ways to modify it to support autonomous design. The next section describes a system that in some ways simulates a human approach, and the subsequent one describes the impact of substantial knowledge infusion on design. Both implementations are for the two-dimensional layout design of urban parks.

3. Approach 1: Collaborative Agents

The program described in this section is consistent with how people solve problems, and how human experts produce designs, in the following ways:

- People solve problems "well enough" rather than optimally (Simon, 1981). Once a solution is in hand, however, it is often improved.
- Designers typically manage classes of objects, rather than individual ones (Goel and Pirolli, 1992). Objects in such a class are functionally and/or structurally similar to one another, such as the fair problem's booths and rides.
- During design, people tolerate states that violate constraints (Schraagen, 1993). In the fair problem, this would be equivalent to searching from a state where two or more structures already overlap. People often rely on such failure-driven conflict resolution (Collins, Birnbaum and Krulwich, 1989). For example, while working on the carousel problem, a design expert might initially overlap the rides slightly, to give her more space in which to manipulate the booths, and reposition the rides later in the process.

FLO (named for Frederick Law Olmstead, a great nineteenth-century designer of urban parks) is a program that casts two-dimensional layout design as a collaboration among a set of intelligent agents (Epstein, 1998). Each agent specializes in a particular kind of object found in an urban park. In the empirical work described here, the object classes were playing fields, buildings, ponds, forests, and roads.

TABLE 1. A park design problem for FLO.

Task: Place on a 20 × 40 discrete grid the following objects:

PF_1: a 7 × 7 playing field	B_1: a 5 × 10 building	R_1: a road of width 1
PF_2: a 5 × 2 playing field	B_2: a 2 × 2 building	R_2: a road of width 1
PF_3: a 5 × 2 playing field	F_1: a 6 × 7 forest	R_3: a road of width 1
P_1: a 5 × 3 pond	F_2: a 4 × 2 forest	R_4: a road of width 1
P_2: a 3 ×2 pond		

Object constraints:

O_1: PF_1 boundary is ≥ 0.1 away from all edges.

O_2:PF_3 center is within 0.8 of the park’s center.

O_3:B_1 boundary is against some edge.

O_4:B_2 center is within 0.5 of the park’s center.

O_5:P_1 boundary is ≥ 0.3 away from all edges.

O_6:P_2 center is within 0.3 of the park’s center.

O_7:F_1 center is within 0.4 of the park’s center.

O_8:F_2 boundary is within 0.1 of some edge.

O_9:R_1 runs from the eastern grid edge.

O_{10}:R_2 runs from the western grid edge.

O_{11}:R_3 runs from the northern grid edge.

O_{12}:R_4 runs from the southern grid edge.

O_{13}:R_1, R_2 , R_3 , and R_4 are connected.

Intra-class principles:

O_{14}: Total road length should be small.

O_{15}: Fields should not adjoin each other.

O_{16}: Ponds’ centers should be ≤ .2 of the grid apart.

O_{17}: Buildings’ centers should be ≤ .6 of the grid apart.

Inter-class constraint:

O_{18}: Objects may not overlap.

Inter-class principles:

O_{19}: Buildings should be adjacent to roads.

O_{20}: Every building should have a view, i.e., be within 2 units of a pond or forest.

Secondary inter-class principle:

O_{21}: Minimize the total Manhattan distance of all empty grid positions to their nearest road.

In the design of a large urban park, like Olmstead’s Central Park in New York, few pre-existing features are retained. (In the following discussion, the references in parentheses are to Table 1, an actual problem given to FLO.) The problem describes *objectives* (constraints and principles) for 13 objects that are not permitted to overlap (O_{18}). If the park is centrally located, vehicular access to the outside world is crucial in all directions (O_9 – O_{12}). These roads should be connected (O_{13}), but the amount of paved surface should remain small (O_{14}). To create a sense of disconnection from the urban world, activity centers should be somewhat distant from the edges of the park itself (O_1) even near the center of the park (O_2). Similarly, environmental features, such as forests and ponds, are generally secluded from the edges of the park (O_5 – O_8,). Some activities are likely to interfere with one another, and so should be separated (O_{15}). Buildings serve various purposes; some are more relevant to the park itself (O_4), while others serve the non-park environment (O_3). Some objects benefit from proximity to others of the same class (O_{16}, O_{17},). Buildings are often used by employees of the park; they therefore require vehicular access (O_{19}) and good views (O_{20}). Finally, in the event of an emergency, no location should be very far from a road (O_{21}).

When a 20×40 integer-based grid is used, the problem in Table 1 becomes challenging; its search space contains about 1.3×10^{26} states. A solution to this problem is any placement of the objects that satisfies all the constraints. If all the objectives in Table 1 were constraints, there is no guarantee that any of those state would be a solution. Instead, enough objectives were designated as principles so that preliminary tests found a solution. Observe too that constraints are easy to test: they are either boolean (e.g., "against some edge") or lie in a numeric range (e.g., "≥ .01 away from…"). Just as in the carousel problem, solutions will differ in their worth. Values provided by the principles all lie in the same range, and principles are assumed to be of equal weight. (In other words, the worth of a state is the sum of its values from the principles, unless the user specifies otherwise. See (Epstein, 1998) for further details.)

Each agent in FLO is responsible for objectives that pertain only to objects of its class (*intra-class objectives*), and is also responsive to objectives that include its class among others (*inter-class objectives*). For example, the building agent must guarantee that the buildings' placements satisfy constraints O_3, O_4, and O_{18}, and also must concern itself with principles O_{17}, O_{19}, and O_{20}.

Rather than seek an optimum, FLO makes multiple attempts to solve the same problem relatively quickly. Each attempt is a three-stage process: generate a start state, repeatedly modify it until all the constraints are satisfied (producing a *solution*), and then tweak the solution to improve its worth. If modification fails to produce a solution within allocated resources,

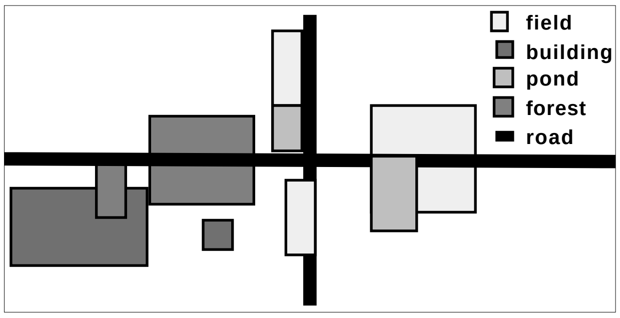


Figure 3. A potential start state for FLO, a combined ideal for the problem in Table 1.

that attempt is abandoned and a new one begun. After some number of attempts, FLO outputs the best of its solutions.

FLO’s agents collaborate to formulate these two-dimensional layout designs. Although the initial state inherent in a two-dimensional layout design problem is an empty grid, FLO devotes a substantial portion of its processing time (21.3% in the runs reported upon here) to the construction of a set of good *start states* from which to begin search. To construct a start state, each agent generates a set of *ideal frameworks*, states that include only objects of its class and abide by all problem constraints on them. Each agent then ranks its ideal frameworks to reflect the degree to which they satisfy its intra-class principles. Next, a set of highly-ranked ideal frameworks, one from each agent, is overlaid to form a *combined ideal.* (Figure 3 is an example of a combined ideal produced by FLO for the problem in Table 1.) It is quite likely that two or more objects from different classes will overlap (produce a *conflict*) in a combined ideal. Therefore, several combined ideals are generated, and in some cases individual ideal frameworks are replaced or modified in a resource-limited attempt to minimize initial conflicts. The combined ideals with the highest worth become the start states.

Once a start state is selected, the second stage iterates to minimize conflicts. In response to the current state, each agent generates a set of constraint-abiding *proposals* (recommended actions). A proposal must either *resolve* (eliminate) or *ameliorate* (reduce the number of objects in) one or more conflicts involving that agent’s objects. In Figure 3, for example, to address the conflict between the lower field and the southern road, the field agent might propose to relocate the field. At the same time, the road agent might propose to bend the road around the field. In any state there are likely to be several conflicts, and an agent may have many ideas as to how those conflicts should be resolved.

Because search is intended to be heuristic and not exhaustive, each agent is permitted to suggest only a few of its best proposals, ranked once again according to the principles known to that agent. In the runs reported here, each agent was limited to at most 10 proposals and at most 10 seconds of computation time. Until a solution is reached or there are no proposals, FLO selects and implements a proposal that addresses the most *severe* conflict, that is, one that involves the most objects, covers the most area, and is the subject of the fewest proposals. If there is more than one such proposal, the one that produces the state with the highest worth is chosen. No proposed action may increase the number or severity of conflicts, nor may it return to a prior state. In each of 10 runs on the problem in Table 1, FLO made 10 attempts to solve any given problem, and succeeded on average 73% of the time, finding a solution in 136.9 seconds. Two solutions produced by FLO for the problem of Table 1 appear in Figure 4.

After a solution is produced, tweaking in the third stage repeatedly tries to improve the worth of the solution state. Once again the agents make proposals which do not introduce conflicts, but this time they also promise to improve the current worth. For example, the solutions in Figure 4 could be improved with respect to O_{20} (good views from buildings). In Figure 4(a), the large building could be shifted slightly upward, while in Figure 4(b) it could be shifted substantially to the right. Tweaking continues until no proposal can make a significant (with respect to some user-determined threshold) improvement on the solution's worth.

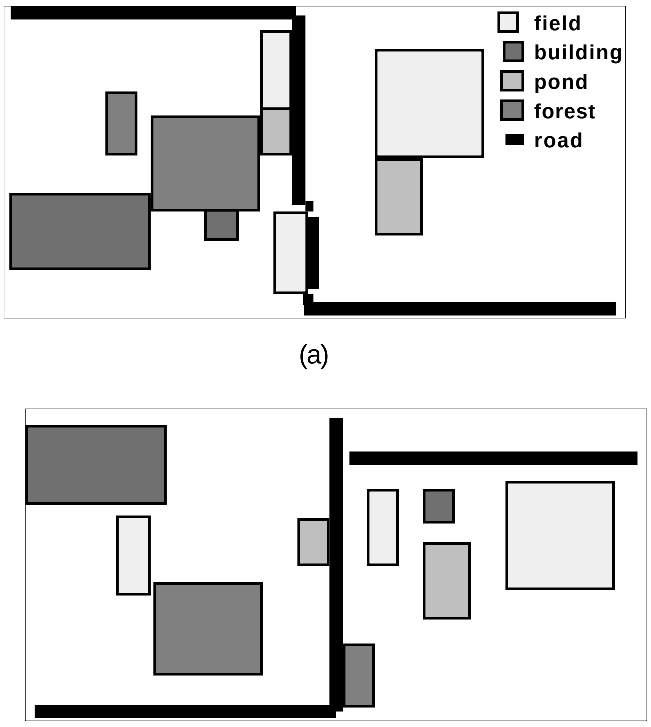
As an AI program, FLO uses a number of novel devices to good effect:

· It searches through a space of *complete* (all objects present), *illegal* (constraint-violating) states, rather than the typical AI backtracking search through a space of partial, legal states. Although FLO's search space is admittedly larger than the set of partial legal states, its carefully selected start states are highly valued by the agents and include relatively few conflicts (and thereby support shorter solution paths). Proposals, too, move toward high-worth states. As a result, FLO usually finds a solution relatively quickly in a large search space.

· FLO distinguishes between constraints and principles. This permits it to find some usable solution with respect to the constraints first, and relegates the vague, optimizing principles to ranking and tie breaking. Once a solution is discovered, however, it is repeatedly refined with respect to those principles.

· The size of the search space is made more manageable by a variety of limitations on the consideration of alternatives. There are limits on the number of ideal frameworks and the number of combined ideals, on the modifications to the combined ideals, on the time to construct proposals, and on the number of proposals generated. FLO demonstrates that relatively few ideas are enough to reach a variety of interesting solutions.

· Routines to manage objects can often be made more efficient and insightful if the objects are in the same class. FLO specializes proposal generation, ideal framework generation, and worth computation to each agent's class. Nonetheless, some principles (called *secondary principles*) are so costly that they should be reserved only for stages 1 and 3. In Table 1, O_{21} is an example of such a principle.



(b)

Figure 4: Two of FLO's solutions to the problem in Table 1.

There are, of course, limitations to FLO's prowess, ones unlikely to appear in the performance of even a novice human designer. FLO does not learn from its own design experience, nor does it employ any available expert design knowledge. FLO relies on chance to find good ideal frameworks, rather than consider successful designs of other parks. FLO disregards preexisting conditions in the area designated for the park, even though they might be relevant to construction. FLO is limited to integer-based coordinates, and all but its roads are expected to be rectangular. There is only a primitive relationship between proposal generation and the conflicts in the current state. The runs on which no solution is found tend to be stalemate situations, where two or more agents would have to simultaneously implement a proposal (e.g., shift two objects of different classes away from each other at once). These limitations motivated the program described in the next section.

4. Approach 2: Guidance from the Problem Space

FLO2 is intended to capitalize on FLO’s successes, and address its limitations. FLO2 is a work in progress, a long-term project to explore a variety of issues in real-world, two-dimensional layout design. In many ways, FLO2 will simulate behaviors recommended to human experts engaged in park planning (Alexander, 1987; Alexander, Ishikawa and Silverstein, 1977; Cullen, 1968; Gottdiener and Lagopoulos, 1986; Hedman and Jaszewski, 1984; Lynch, 1971; Lynch, 1981; Molnar and Rutledge, 1986; Moughtin, 1996; Peps, 2000; Stephen-Cutler and Stephen-Cutler, 1983). At this writing, the initial phase (described below as a sketch) is implemented, as well as one costly, complex secondary principle. FLO2 retains FLO’s object-class orientation and its distinction between constraints and principles, as well as its multiple-attempt approach. The author is a member of the team involved in the FLO2 project.

The differences between FLO and FLO2 are motivated primarily by two intentions: to anchor design in real-world data and to give the designer more discretion. These differences include:

· The context of an urban park is a city, which offers transportation and utilities at the edges of the park, and perhaps inside it. Instead of an empty grid, FLO2 states are described with respect to a *design frame* that specifies both the *site* (where the objects are to be placed) and the *periphery* immediately adjacent to it. An example of a design frame appears in Figure 5. The design frame includes relevant pre-existing objects, such as utility accesses and bodies of water. The frame/periphery distinction is fully implemented.

· Often a designer has some ability to determine the size, shape, and orientation (with respect to climate and light) of a park’s objects. Instead of a fixed-size rectangle, each FLO2 object has a set of *property values* that varies in its degree of specificity and flexibility. A soccer field, for example, would have fixed dimensions, while a picnic area would be more flexible. In a solution, the system *anchors* every object, assigning it fixed dimensions, a shape, a location, and an orientation. The property value approach is implemented for certain objects; the remainder is under construction.

- The most computationally complex objects in FLO are the roads, and there are other things to transport in a park besides vehicles. Instead of FLO's individual road segments, FLO2 supports a broader category of objects called *container networks.* A container network is a connected graph intended to provide transport or access, such as hiking trails or power lines. A container network consists of a *skeleton* (its principal arteries) and *branches* (connectors to specific locations). For example, Olmstead's typical road skeleton for an urban park was a centrally-located ellipse. In a particular park, there could be a branch from that skeleton to the riding stable, and another to the restaurant. Container networks are currently under construction.
- Human designers are taught to develop layout designs at different levels of detail, beginning with rough forms and eventually adding detail to their best ideas. In addition to objects, FLO2 requires that design specifications include *use areas,* sections of the park intended to support particular kinds of activities. Kinds of use areas include playgrounds, sports fields, water-related activities, entertainment, forests, meadows, food service, maintenance facilities, and container network skeletons. There may be more than one use area of the same kind in a single park. Use areas are fully implemented.

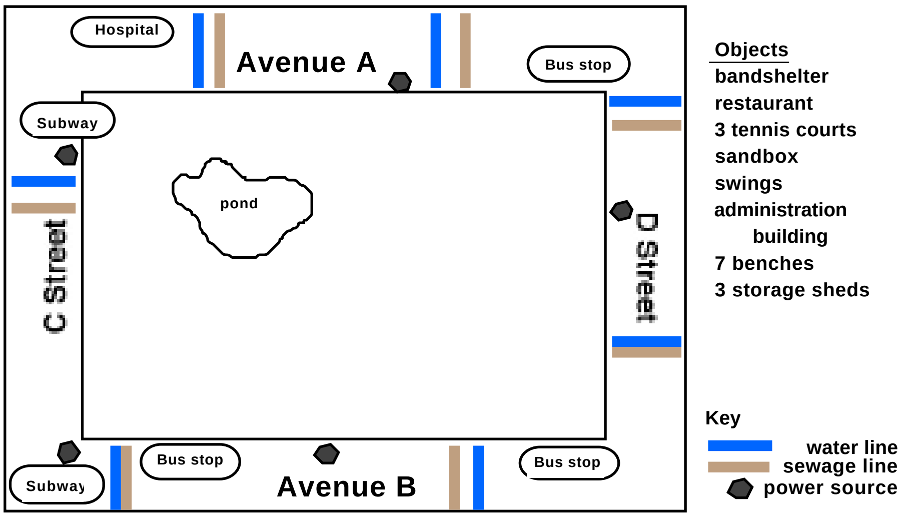


Figure 5. A design frame.

TABLE 2. Objects for the problem of Figure 5.

Use area	Type	Objects
Maintenance	major	Administration building
Entertainment	major	Band shelter, restaurant
Playground	major	Sandbox, swings
Playground	minor	Bench 1, bench 2, bench 3
Playing fields	major	Tennis court 1, tennis court 2, tennis court 3
Playing fields	minor	Bench 4, bench 5
any	minor	Bench 6, bench 7
any	minor	Shed 1, shed 2, shed 3

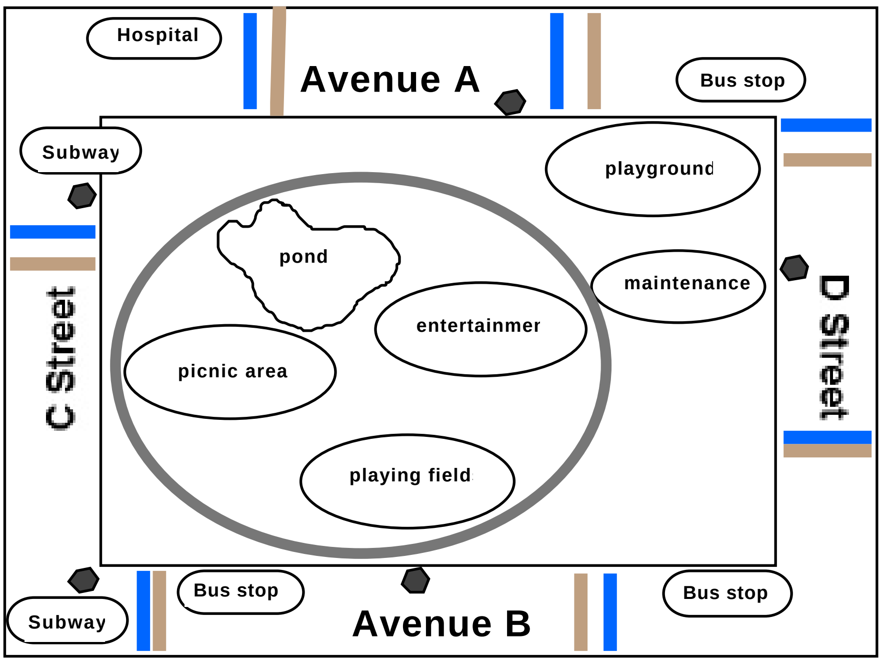


Figure 6. A sketch for the design frame of Figure 5.

In addition to object classes, FLO2 requires the user to distinguish between *major objects* (ones that have a significant impact because of their size and use) and *minor objects* (all others). A restaurant or a sandbox could be a major object because it attracts many park visitors; a bench or a storage shed is likely to be a minor object. Furthermore, each major object must be designated for some use area. An example for the design frame of Figure 5 appears in Table 2. Object categorization is under construction.

· Instead of illegal complete states, start states are knowledge-rich, legal abstractions of a design. Search moves first among legal abstractions, and later refines such abstractions to produce a design. Start states are fully implemented.

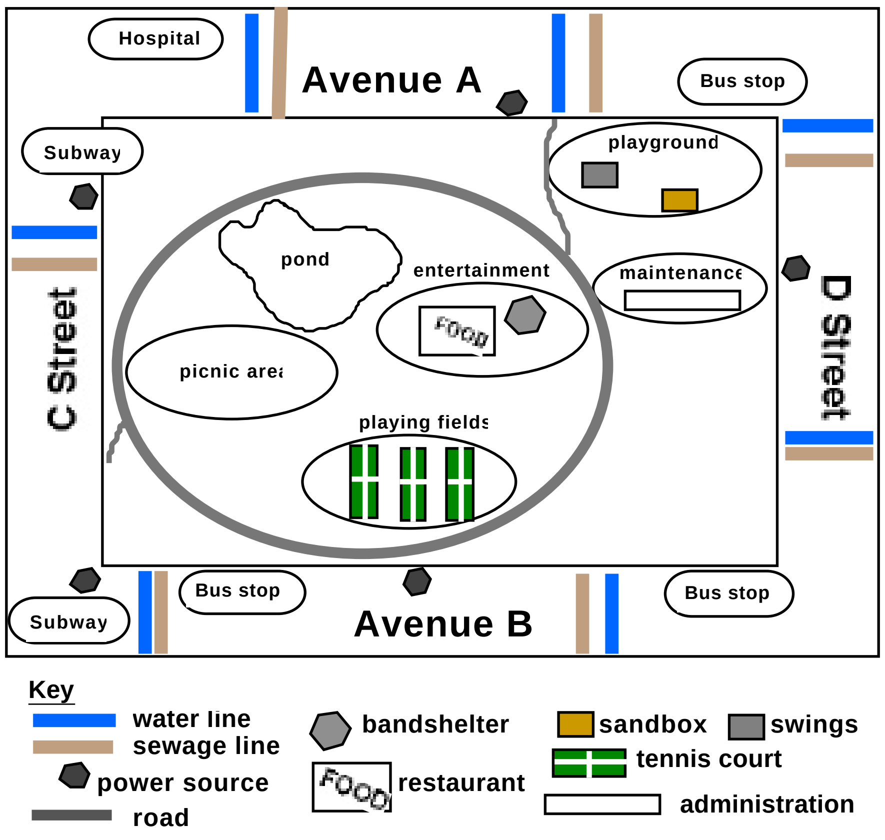


Figure 7. A plan for the sketch of Figure 6.

Because FLO2 must shape and orient its objects, its search space is even larger than FLO's. FLO2 therefore makes use of two well-known AI devices to make search more manageable: it searches at increasing *granularity* (level of detail), and it informs search with knowledge. Instead of a single search, FLO2 requires three searches at increasing levels of detail to produce one design.

First, FLO2 produces a *sketch*, which anchors only use areas within the design frame. A sketch permits the designer to ignore details and focus instead on pre-existing terrain features. This high-level perspective can "tune" a layout to its design frame, both aesthetically and economically. A sketch is an abstraction of a design; an example for the design problem of Table 2 and Figure 5 appears in Figure 6. Note that the road network appears as a skeleton here. Except for networks, the sketch phase is fully implemented.

Next, FLO2 anchors all major objects and appends all network branches to produce a *plan*. At this stage, deferring less pressing detail permits the design process to focus on the most important principles. An example of a plan for the sketch of Figure 6 appears in Figure 7. The plan phase is under construction.

Finally, FLO2 anchors all minor items to produce a *design,* and then tweaks that design much the way FLO does. An example of a design for the plan of Figure 7 appears in Figure 8. This third stage is also under construction.

FLO2 relies on two kinds of knowledge to produce good start states for a sketch: terrain constraints and sketches of existing parks. The nature of the underlying terrain impacts upon both the practicality of park construction and its cost. *Terrain constraints* describe the nature of the land enclosed by the design frame: its slope, drainage, vegetation, and soil type. Sets of values for these constraints designate *property types*, and kinds of construction are deemed more or less suitable for a particular property type. For example, land with good or fair drainage, good or fair soil, slope no more than 4%, and poor vegetation is categorized as type IIA, and acceptable for minimal construction. Terrain constraints have been codified as a reference manual for landscape architects (Molnar and Rutledge, 1986); that data is fully accessible to FLO2. Using terrain survey data provided by the user for the design frame, FLO2 constructs an *acceptance map* (a coded two-dimensional representation of how appropriate each grid cell in the design frame is) for each property type. In addition, because well-respected design layouts are likely to embody good solutions to recurrent issues, FLO2 has a *case base* of park sketches, extracted from a set of real-world urban parks.

Initially, the user specifies the use areas for the park to be designed, as well as constraints on the distance between any pair of use areas. Such a pairwise constraint is either an *attraction* that specifies maximum distance (e.g., "keep the camping area within 100 feet of the picnic area") or a *repulsion* that specifies minimum distance (e.g., "keep the playing fields at least 500 feet from the playground.") Each constraint appears in a FLO2 sketch as a link between its use areas; repulsions can stretch during search through possible sketches, and attractions can contract. Next, the user mandates a start state for the production of a sketch, or has FLO2 suggest similar cases from its case base as start states. FLO2 measures the worth of a sketch as the average values of the use areas based on the acceptance maps, and searches by re-anchoring the use areas to improve that worth. (The user can also override FLO2, and do this with the mouse.) During this search, the use areas shift and pivot about their links. An example of a sketch produced by FLO2 appears in Figure 9.

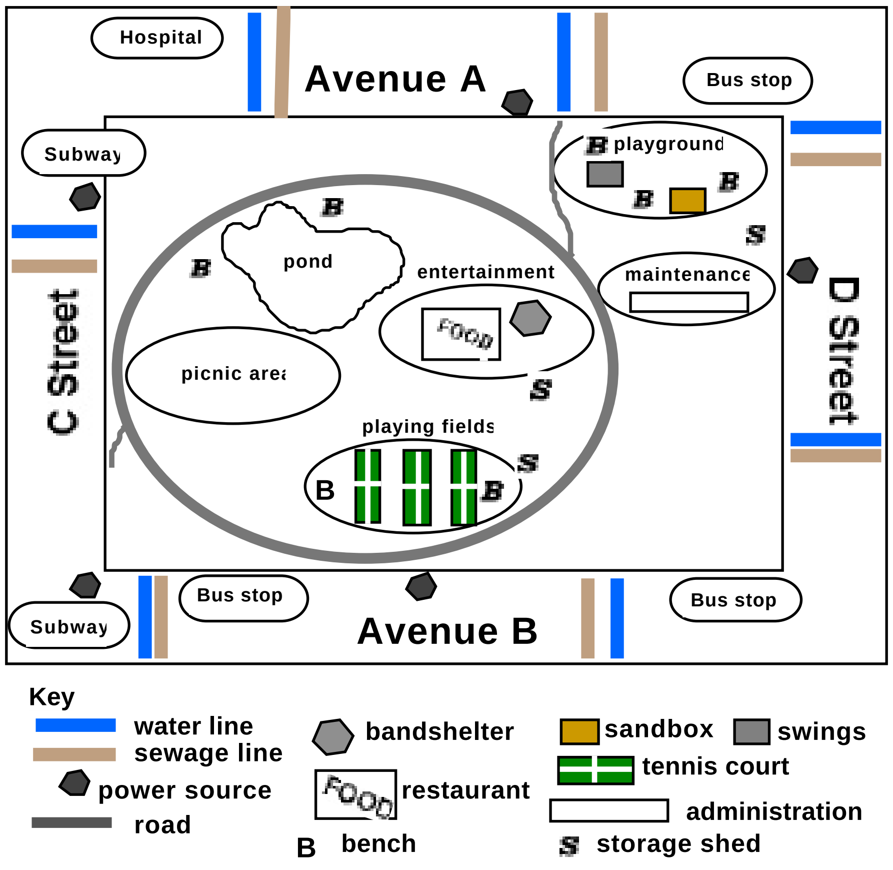


Figure 8. A design for the plan of Figure 7.

Implementation of FLO2 is well underway. Plans will be created separately for each use area, a divide and conquer approach well-known for its efficiency in algorithmics. If the user requests it, a library of cases for use areas (e.g., good playgrounds, good picnic areas) will provide start states for search. A plan for the park itself will be a set of plans for the use areas, and high-worth plans will be combined to form a full plan, much the way ideal frameworks in FLO created a combined ideal — in FLO2, however, there will be no possibility of overlap, since use areas are separate.

FLO2 uses knowledge not only to select a good start state, but also to select a good next state during search, one that should rank high on principles. For example, FLO2 simulates a secondary principle called *navigability* that considers how easily visitors can travel within the park. A simulation moves hundreds of simple, reactive agents through a design in real time as if they were members of a particular class (e.g., toddlers, hikers). Each class determines its agents behavior: speed, length of visit, target destinations, even their willingness to remain on a path. As they leave the park, each agent provides information on its experience with respect to features such as crowding and successful arrival at its targets. That data, along with other observations on traffic and congestion gathered during the simulation, can suggest modifications to a design. If, for example, the children found the restroom facility too far from the playground, or the hikers found themselves on overly-populated trails, the design could be tweaked to improve its worth based on those reports. Thus tweaking in FLO2 is simulation-guided, rather than fortuitous.

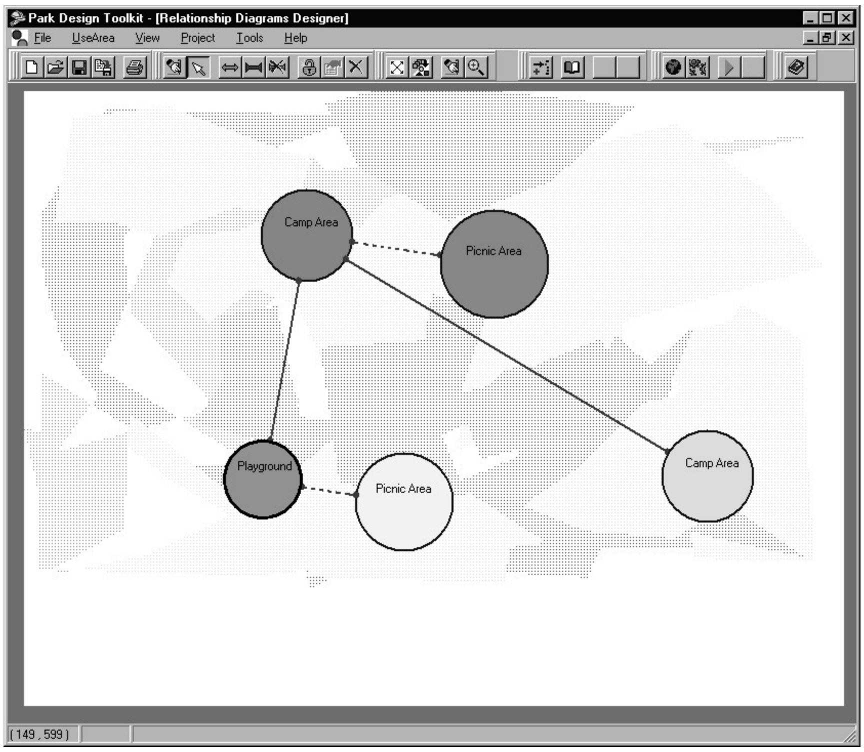


Figure 9. A sketch generated by FLO2. The links are use area distance constraints.

5. Future Work

FLO2 is still under construction. Current work is directed to the construction of a plan case base, and automation of the progression from sketch to plan to design. Because principles impact the quality of the final design, good (i.e., properly descriptive) metrics on them are essential. Aside from the obvious difficulties in quantifying aesthetic principles, FLO2 requires that such computation be relatively quick. It is possible, however, to include two versions of a principle, if need be — a fast one plus a secondary-principle version (like that of navigability above). In addition, tweaking will eventually include gradual changes in the shape of specified objects, so that individual agents may move their objects toward forms that the agents consider more worthy.

Moreover, real experts learn. Future work includes the identification what (beyond cases for its libraries) FLO2 should learn, and the construction of algorithms to acquire and apply such knowledge. Possibilities include the sequence in which objects should be placed, relative importance of principles observed in decision sequences from human designers, and the ways particularly successful tweaking might be implemented as additional principles instead.

6. Lessons for Autonomous Design

Although design problems are ill-structured and entail enormous search spaces, it is possible for an AI system to construct interesting and successful designs on its own. FLO and FLO2 take somewhat different approaches, but they demonstrate several important lessons that should be applicable beyond parks, and even beyond two-dimensional layouts, for those seeking autonomous design programs:

- A good start state is important. Devote substantial resources (both on-line computation and off-line information gathering) to its selection.
- Given enough good start states, variety is readily produced by repeated solution attempts. Decisions that are nearly the best lead to surprisingly varied and valuable outcomes.
- Much specific knowledge is actually limited to categories of objects, and is best applied there. Identify features of objects that lead to good design, and use those features to guide search once it begins.
- Most design goals are really principles. Distinguish carefully between the few properties that are required and those that are merely desirable.
- Knowledge from both the particular problem and related problems is essential to expertise. Embed it both in the formulation of the start state and as often as possible during search.

- Good designs can often be improved. Concentrate first on getting one that works, and tweak the details later.

Acknowledgements

Many thanks to those who believed in FLO enough to devote their considerable wisdom and efforts to FLO2: Janice Glasgow, Bernard Moulin, Geoffrey Edwards, Gérard Ligozat, Tony Marley, Walid Chaker, and Jeremi Gancet. FLO was supported in part by NSF #9423085 and by PSC-CUNY #666318. FLO2 is supported by the GÉOIDE project of Canada.

References

Alexander, C.: 1987, *A New Theory of Urban Design*, Oxford University Press,

Alexander, C., Ishikawa, S. and Silverstein, M.: 1977, *A Pattern Language: Towns, Buildings, Construction*, Oxford University Press, New York.

Collins, G., Birnbaum, L. and Krulwich, B.: 1989, An adaptive model of decision-making in planning, *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Morgan Kaufmann, pp. 511-516.

Cullen, G.: 1968, *Townscape*, Reinhold.

D'Andrade, R. G.: 1990, Some propositions about the relations between culture and human cognition, *in* J. W. Stigler, R. A. Shweder and G. Herdt (eds), *Cultural Psychology: Essays on Comparative Human Development*, Cambridge University Press, Cambridge, pp. 65-129.

D'Andrade, R. G.: 1991, Culturally based reasoning, *in* A. Gellatly and D. Rogers (eds), *Cognition and Social Worlds*, Clarendon Press, Oxford.

Epstein, S. L.: 1998, Toward design as collaboration, *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, Madison, WI: AAAI, pp. 135-142.

Goel, V. and Pirolli, P.: 1992, The structure of design problem spaces, *Cognitive Science*, **16:** 395-429.

Gottdiener, M. and Lagopoulos, A.: 1986. *The City and the Sign: An Introduction to Urban Semiotics*. New York: Columbia University Press,

Hedman, R. and Jaszewski, A.: 1984, *Fundamentals of Urban Design*, Planners Press, Washington, D.C.

Lynch, K.: 1971, *What Time is This Place?*, MIT Press, Cambridge, MA.

Lynch, K.: 1981, *Good City Form*, MIT Press, Cambridge, MA.

Molnar, D. J. and Rutledge, A. J.: 1986, *Anatomy of a Park*, McGraw Hill,

Moughtin, C.: 1996, *Urban Design: Green Dimensions*, Butterworth-Heinemann, Oxford.

Peps, J. W.: 2000. *1794-1918: An International Anthology of Articles, Conference Papers, and Reports selected, Edited, and Provided with Headnotes*, Cornell University,

Schraagen, J. M.: 1993, How experts solve a novel problem in experimental design. *Cognitive Science,* **17**(2)**:** 285-309.

Simon, H. A.: 1981, *The Sciences of the Artificial*(second ed.), MIT Press, Cambridge, MA.

Stephen-Cutler, L. and Stephen-Cutler, S.: 1983, *Recycling Cities for People - The Urban Design Process*, Van Nostrand Reinhold, New York.