制約に基づくフロアプランニング自動化システムの実現方法

本多一賀†

本論では，フロアプランニングを自動化するための制約を中心としたモデル化と，制約の効率的な処理を可能にしたシステムの実現方法を提案する．我々の方法は，配置対象と配置対象物間の関係を「制約」で表現し，この「制約」を効率的に処理するシステムを実現することでフロアプランニング，主に初期設計段階の第一次案の作成を自動化する．この方法は，(1) 対象を宣言的な制約として記述するのでモデルがつくりやすい，(2) モデル化とモデルを解くための処理系が独立しているので，プログラミングの拡張性，保守性が良い，という利点を持つ．実現にあたっては，(1) 制約処理に関しての並列処理（コルーチン処理技術）と制約伝播を活用した効率の良いアルゴリズム，(2) フロアプランニングの自動化に必要な数値計算と記号処理の効果的な融合を行うための有限領域を対象とした制約論理型言語の枠組み，の２つを導入しており，本稿では，これらについて示す．

Constraint-based Approach for Automatic Floor Planning System

KAZUYOSHI HONDA†

We present the new methods to realize the automatic floor plannig system that introduce constraint-based modeling and constraint processing. Our new method represents the objects and its rclations as constraint first, and after that, it draw the layout based on our efficient constraint processing mechanism that are independent on the domain itself. The main target of our methods is to make the initial plan. Using our methods, we got the merits such as (1) delrarative description of domain model and (2) high extensibility and low cost maintenance of the system. Our method was realized by using (1) efficient algorithm using constraint propagation based on co-routine mechanism in order to solve constraint and (2) constraint logic programming framework to combine the arithmetic and symbolic calculation to solve the automatic floor planning problem. In this paper, wc cxplain thcsc mcchanism in detail.

1. はじめに

住宅間取り作成[14]，オフィスビル内での会社，部課所単位でのエリアの割振り[3]など，二次元のレイアウトの設計作業は，フロアプランニングと呼ばれる．手間がかかることと頻度が多いという理由から，主に初期設計段階の第一次案設計作業を自動化するシステムの作成が試みられた．

システム実現のアプローチとしては，配置対象の部屋を隣接関係に基づくグラフを利用して配置を行うものと，複数のルールを使って配置するものがある．いずれの方法も，フロアプランニング問題が組合せ問題であるという特徴を考慮して，組合せの爆発を回避した効率的な解の導出を目指している．

前者の例としては，Hashimshony ら[6]の方法がある．今，a，b，c，dの４つの部屋があり，aとb，bとc，cとd，aとdに隣接関係が与えられているとする．このとき，**図1**のように，与えられた隣接関係に基づいて，図１中の (1) のように隣接グラフを生成し，さらに，図１中の (2) のように実際のレイアウトを導出するために各部屋の方位の決まった平面グラフを生成して[10]，(3) に示すレイアウトを得る方法である．

この方法を実際の問題に適用する場合，レイアウトを行うのに必要十分で，しかも矛盾のない隣接関係を与えること[6]や，隣接関係と方位以外の部屋の大きさも考慮したレイアウトを生成することが難しい[10]という問題がある．

これに対して，隣接関係による配置にこだわらず，配置のルールを利用してレイアウトの導出を行う方法が Pfefferkorn[14]，Flemming[4]らによって提案された．この方法は，基準となる部屋や壁を設定して，この基準物との制約（隣接，方位，隣接させない，など）に基づくルールによって，部屋を配置していく．たとえば，図１と同じ問題を，**図2**のようにして解く．図２の Step2 では，aがおかれると「隣接する

† 東京ガス株式会社
Tokyo Gas Co., Ltd.

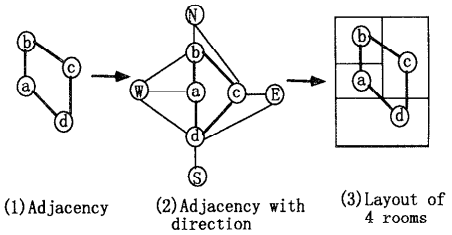


図1 グラフを利用した配置
Fig. 1 Spatial layout based on adjacency graph.

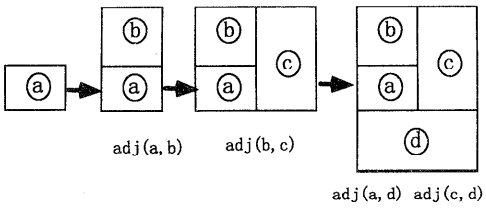


図2 配置ルールを利用した配置
Fig. 2 Spatial layout based on the rule.

部屋を配置する」というルールによってbを配置し，Step3では，同じルールによって，bと隣接するcが配置される．さらに，Step4で，aとd，cとdとの隣接関係によってdが配置される．

以上のルールベースを使った方法で発生する問題はプランニング問題で発生するルールの競合である．たとえば図2のケースでは，Step2においては，bではなく，同じく隣接関係にあるdを配置することも可能であり，Step3でも，cを配置せずに，先にdを配置することも可能である．これらすべての場合を試行することは組合せの爆発をまねくことになる．

そこで競合の発生しない，より詳細な条件のついたルールを導入することや，競合解消のためのメタルールの導入が行われた．たとえば同じ隣接でも，すでに配置されているものや，空領域の状況によって，小さいものから先におくことや，隣接関係の多いものから先におくことなどをルール化する．しかしながら，配置対象の形状，配置物の大きさ，位置等，与えられる制約条件に依存したルールは数が多く，ルール間の整合性をとることが難しいという問題があった[1]．

これらの従来のアプローチに対して，我々が試みているのはグラフやルールではなく，対象と領域の知識を方程式と論理式からなる制約で表現し，この制約を処理することで配置を実現する方法である．

この方法は，対象に依存した細かなルールの記述やグラフに頼るのではなく，問題に含まれる制約を解くシステムを実現することで解を導出する．このため，対象の記述を容易にできるという利点と，システムを実現するためのプログラミングの観点からの拡張性，保守性が良い，という利点が得られる．

しかし，問題を制約で表現して，既存の制約処理システム[2]で解かせるだけでは効率が悪く，実用的な時間で解を得ることは困難である．

そこで，制約処理（具体的には候補の生成と絞り込み）に関しては，部分的に並列処理（コルーチン処理）を導入したアルゴリズムの開発や，制約処理効率の向上に有効であることが知られている配置順序等の最小限度の経験則[5]を導入することで，実用的な時間での解の導出をめざした．

実装にあたっては，ここ数年で飛躍的に進歩した，有限領域を対象とした制約論理型言語[2),11]による定式化，プログラムレベルでの並列プログラミング技術を活用した．

2章では我々の取り扱うフロアプランニング問題のクラスを定義し，現実問題に含まれる問題の特徴についても説明する．

3章では住宅間取作成の例を導入し，我々の考えるフロアプランニング問題の「制約での宣言的な表現方法」について示す．

さらに4章では，モデル化とは独立した，制約を解くためのアルゴリズムを示す．

5章では，4章のアルゴリズムをもとに，我々が作成した住宅間取システム，TG-FP（Tokyo Gas Floor Planner）の例を導入して有効性を示すとともに，アルゴリズムの効率についても議論する．

最後に，結論と今後の課題を述べる．

2. フロアプランニング問題

2.1 フロアプランニング問題

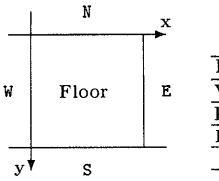
フロアプランニング問題は，(1) 部屋，エリア，セクションなどのオブジェクトと，(2) 敷地，部屋，ビルの一階分のフロアなどの配置対象領域，そして (3) 配置のための制約条件が与えられる．フロアプランニング問題の解は，必要なオブジェクトを，以下のような制約条件[14]に従って，配置対象領域に配置することによって得られる．

(1) 距離 （オフィスプランで）経理部と営業部は10 m以内に配置する．

(2) 位置 （オフィスプランで）備品収納エリアは，フロアの一番すみに配置する．

(3) 配置方向 （住宅設計で）ガレージは，車を道路に並行におくようにつくる．

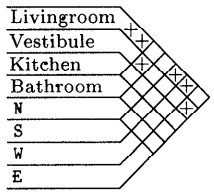
(4) 隣接 （オフィスプランで）経理部と営業部は隣接させる．



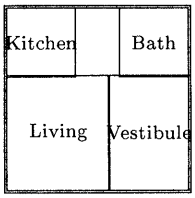
(a)Floor

	Depth		Width	
	min	max	min	max
Livingroom	360		360	
Vestibule	120	600	120	600
Kitchen	180		180	
Bathroom	180		180	
				700

(b)Size



(c)Adjacency



(d)PlanSample

図3 例題1
Fig. 3 Example of floorplaning problem.

(5) 空間的 （オフィスプランで）2つのエリアは，重ならない．

(6) 通路 （住宅設計で）すべての部屋からドアへの通路がある．

取り扱う問題によってこれらの一部または，すべてを含むことになる．

フロアプランニング問題の一例[1]を以下に示す．

例題1

部屋を配置する領域 *Floor*（図3(a)），部屋の大きさの制約 *Size*（図3(b)），隣接関係の制約 *Adjacency*（図3(c)）が与えられている．ここでは制約としては，「Livingroom の大きさは横幅が最小 360，縦幅が最小 360」，「Livingroom と Kitchen は隣接する」，「LivingRoom は南向きに配置する」などの条件を与える．先の例では，隣接，空間的，位置に相当する制約が与えられている．

これらの制約条件を満足するように位置や大きさを決定して，解として *PlanSample*（図3(d)）に示す間取りを得るのが問題である．

2.2 フロアプランニング問題の特徴

設計者の観点からの「フロアプランニング問題の特徴」[1),4)]は以下である．

(c1) 解を取り扱える程度の数に制限するのに十分な数の制約がない．

(c2) 与えられる制約は矛盾しやすく，矛盾していればすべての制約を満足する解はない．

(c3) 設計（過程）は，設計者が追加する制約を見つけることと制約を緩和することである．

(c4) 設計者は，問題の定義に暗に含まれる可能性とトレードオフを評価するために，きちんと意味のある違いを持つ解の集合を探す必要がある．複数の設計案が欲しい場合，ポイントとなる部分以外が少し違った案を作っても意味がない．

(c5) 制約を変更することによって，問題の難しさを減少させ，取り扱える範囲に解の数を限定し，求解のプロセスを単純化させる必要がある．

このような性質のために，設計者にとってもフロアプランニング問題は解きにくい問題となっている．これらをシステム実現の観点で見ると，以下の問題の解決を図ることになる．

(s1) アンダーコンストレイント 答はたくさん得られるが，与えるべき制約が不足していて，本当に欲しい答が得られない．

(s2) オーバーコンストレイント 与えられた制約どうしが矛盾していて，答が得られない[9)]．

(s3) 領域知識の効果的な適用 (s1)，(s2) を回避する方法の1つとして，いかに領域の知識を獲得，適用するか．

(s4) 組合せ的爆発の回避 与えられた問題の制約が厳しくても，実用的な時間でいかに答を得るか．

すなわち，フロアプランニング問題は，与えられた問題の制約だけで解く性質のものではなく，システム側で，制約条件の追加削除を行う必要がある．しかしそのことによって，組合せ的な爆発を招くことや，答がなくなってしまうことを回避する必要がある．また，現実的でない答を導出することも避ける必要がある．

3. 制約に基づくフロアプランニング問題の解法

我々は，前章の (s1) から (s4) までの特質を考慮して，フロアプランニング問題を，(1) 前提条件の設定，(2) 制約による問題の表現，(3) 領域の知識の追加，(4) 制約処理による抽象的な配置，(5) 値の決定，というプロセスで解くことを提案する．我々の方法の特徴は，(2)，(3) で，問題を数式と論理式からなる制約で宣言的に記述し，これらの制約処理を効率的に行う (4) のステップで解を得ることにある．

以下では，始めに，例題1を使って (1)～(3) の主にモデル化の部分を説明し，その後，制約処理に対応した (4)，(5) については，説明を分かりやすくするため，他の例も導入して，我々の方法について示す．

3.1 (1) 前提条件の設定

始めに，与えられたフロアプランニング問題を以下の前提条件にあうように修正する．これらの前提条件は制約処理を容易にし，システムを実現しやすくするためである．しかし，実際の住宅間取作成やオフィス

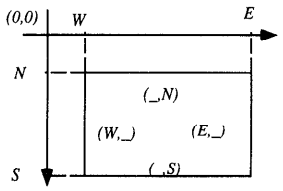


図4 部屋の空間的表現
Fig. 4 Spatial representation of the room.

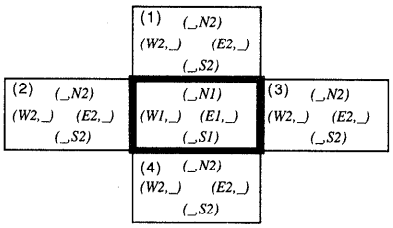


図5 部屋の隣接関係
Fig. 5 adjacency.

プランなどを想定した場合でも，(1) の前提は，施工のコスト等や，配置領域の有効活用の点から，(2) についても，多層階のオフィス，集合住宅においても，一階ごとに設計していくという点から，現実問題に則した前提条件であると考えられる．

(1) 部屋や配置エリアも斜めや R を含まない

(2) 三次元的な配置は考えない

以下の説明では，これらの前提条件を満足している例題 1 を利用する．

3.2 制約による問題の表現

ここでは，与えられた問題を表現するための空間（的な）制約について述べる．

3.2.1 空間制約の表現

空間制約は以下のように定義できる．

部屋：（部屋名，位置，大きさ）

位置：パラメータ $[W, E, N, S]$（**図 4** 参照．たとえば部屋の北側の壁の情報は，視点 W，終点 E，Y 座標 N として表現されている）

大きさ：部屋の幅（Width）と奥行（Depth）は $Width = E - W$，$Depth = S - N$ と定義される．また，以下のようにそれぞれ上下限値を持つ．

$$LowerWidth \leq Width \leq UpperWidth,$$
$$LowerDepth \leq Depth \leq UpperDepth$$

単位：長さの単位は，工法に依存した最小単位を利用する．たとえば木造住宅では，910 mm の 1/3（約 300 mm），910 mm の 1/2（約 450 mm）の単位を扱う必要がある[13]ので，基準寸法 910 mm の 1/6 を最小単位に設定する．

配置対象領域：（西側の壁の集合，東側の壁の集合，北側の壁の集合，南側の壁の集合）

たとえば，北側の壁の集合：$[N_0, N_1, \ldots, N_n]$ 座標軸は部屋と同じ

制約条件：以下の種類の制約が定義できる

方位：方位（部屋）

方位の領域 [north, south, east, west, northwest, northeast, southwest, southeast]

最低限度の面積（方向）：いくつかの場合に，最小限度の面積を設定する必要がある．それは，部屋の幅と奥行が以下のように定義されるとき，$LowerWidth \leq Width \leq UpperWidth$, $LowerDepth \leq Depth \leq UpperDepth$，部屋の大きさが $LowerWidth$ と $LowerDepth$ になるのを防ぐ必要があるためで，以下のような関係を導入する．

$$Width \times Depth \geq MinimumAreaOfRoom$$

隣接関係（adjacent）：2 つの部屋の隣接関係は**図 5** のように，以下の 4 つの関係で定義できる（ここでは，Room1: $[W1, E1, N1, S1]$ と Room2: $[W2, E2, N2, S2]$ を考える）．定義中の番号は図 5 に対応している．関係のうちの 1 つでも成り立てば隣接しているといえ，最後の定義はこれに対応している．

(1) north adjacent(NAj):
$N1 = S2 \wedge E1 \geq W2 \wedge E2 \geq W1$

(2) west adjacent(WAj):
$W1 = E2 \wedge S1 \geq N2 \wedge S2 \geq N1$

(3) east adjacent(EAj):
$E1 = W2 \wedge S2 \geq N1 \wedge S1 \geq N2$

(4) south adjacent(SAj):
$S1 = N2 \wedge E2 \geq W1 \wedge E1 \geq W2$

Adjacent(adj): NAj $\vee$ SAj $\vee$ EAj $\vee$ WAj

重ならない（nonoverlap）：2 つの部屋が重ならない関係は以下のように定義できる（ここでも，Room1: $[W1, E1, N1, S1]$ と Room2 を考える）．このうちのどれかが成立すれば重ならない関係である．番号は図 5 の位置関係と同じであるが，共有部分を持たない．

(0) Room1 と Room2 が隣接関係（adjacent）にある

(1) Above nonoverlap (ANC): $S2 > N1$

(2) Left nonoverlap (LNC): $E2 > W1$

(3) Right nonoverlap (RNC): $W2 > E1$

(4) Below nonoverlap (BNC): $N2 > S1$

以上の定義に基づき，例題 1 を表現すると以下になる（一部，省略してある）．

$adj(Living, Vest.) \wedge adj(Living, Kitchen) \wedge$
$\ldots$(隣接)
$\wedge 360 \leq width(Living) \wedge \ldots$(大きさ)
$\wedge nonoverlap(Living, Vesti.) \wedge \ldots$(重ならない)
$\wedge east(Vesti.) \wedge southwest(LivingRoom) \ldots$(方位)

3.2.2 (3) 領域の知識の追加

領域の知識の追加は，2.2節に示したフロアプランニング問題の性質である，(1) 対象領域を表すのに十分な制約が与えられず，得られる解が妥当なものにならない，アンダコンストレイントのケース，(2) ユーザから与えられた制約は矛盾があり答が得られないオーバコンストレイントのケース，に対応するために行う．

前節に示した制約を必要に応じて，もとの制約知識に付加する．

問題に含まれる部屋に関して，方位の制約がまったく与えられなければ，デフォルト値として，いくつか可能な方位を与える．与えられていたとしても，この制約では問題が解けないこともありうるので，その他の候補を与える．

部屋単独ではなく，複数の部屋の方位を組み合わせて与える場合も多い．住宅の間取では，リビング，ダイニング，キッチンの関係や，風呂と洗面所の関係がこれにあたる．

たとえば例題のケースでは，知識として，LivingRoomが南東でkitchenが東という知識を組み合わせて与えることにすると，以下のような定式化が行える．

$(adj(Living, Vesti. \wedge adj(Living, Kitchen) \wedge$
$\ldots$(隣接)
$\wedge 360 \leq width(Living) \wedge 360 \ldots$(大きさ)
$\wedge nonoverlap(Living, Vesti.) \wedge \ldots$(重ならない)
$\wedge east(Vesti.) \wedge (southwest(Living) \ldots$(方位)
$\vee (southeast(Living) \wedge east(Kitchen)))$
$\ldots$(領域知識の追加)

4. 制約処理による抽象的レイアウトの作成

式(2)のように定式化された問題を，含まれる制約を処理することで解く．始めに，専門家が初期設計で作るエスキースとよばれる抽象的なレイアウト（**図6**）を導出する[13]．円は扱いにくいので，大きさの決まっていない矩形で表現する（**図7**）．

自動設計のアプローチとしては，始めから部屋の大きさと位置を決めているものが多いのに対して[12]，我々は初期段階では，大きさと位置を抽象的に決定する方法をとる．サイズの違いを吸収した形での意味のある違いを持つレイアウトを導出するためである．

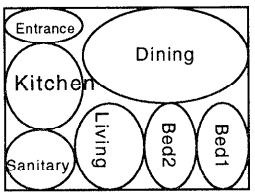


図6 エスキース
Fig. 6 Sketch.

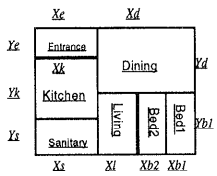


図7 エスキースの矩形表現
Fig. 7 Rectangle representation of Sketch.

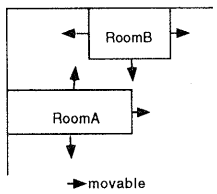


図8 部屋の配置：我々の方法
Fig. 8 Put rooms: our methods.

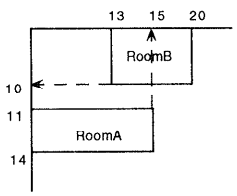


図9 部屋の配置：従来の方法
Fig. 9 Another methods.

具体的には，配置において与えられた制約条件で，答が1つに決まるまで，部屋の大きさと位置は固定しない．我々のアプローチ（**図8**参照）は，大きさを固定する従来のアプローチ[14]（**図9**参照）と違って，どの壁に配置するかと，他の部屋との相対的な位置関係だけを不等式を使って決めている．部屋は，これらの制約を満足する範囲では自由に移動できるので，細かな位置やサイズの問題で配置に失敗することがない．

しかしながら，この特徴だけでは，効率的にレイアウトを導出することは困難である．

それは計算量が膨大になるケースが発生するためである．たとえば，W 個の壁で囲まれる配置領域に n 部屋を並べる場合，最悪の場合，計算量は $W^n 4^n {}_{C_2}$ になる（ここでの4は，「重ならない」の定義における4方向のOR関係を示す）からである．

以下では，この計算量の問題を考慮した効率の良い配置アルゴリズムを示すが，実装言語に依存する部分があるので，始めに実装言語である制約論理型言語[2]について簡単に説明する．

次に，コルーチン機能と数式処理を組み合わせることで実現した効率の良いアルゴリズムを説明する．

4.1 制約論理型言語

我々が実装に利用した制約論理型言語ECLIPSE[11]は，ECRCで開発された言語である．有限領域での線形不等式・等式を改良シンプレックス法で解くことができる．また，効率的な探索のためのコルーチン制御機能を持っている．

言語のシンタックスは，Prologとほぼ同じある．違

いは，探索方法として，Prologの縦型探索に加えて，分枝限定法も利用できること，等式の表現と変数，変数の値の遅延が行えること，変数に領域を持てることである．

変数について説明すると，数値の場合，変数 :: 下限値.. 上限値（例：X :: 10..20）のように表現する．領域における演算も行われ，たとえば，X :: 10..20, Y :: 15..20を与え，X #> Y なら，X :: 16..20, Y :: 15..20のように計算される．記号の場合，変数 :: 候補（例：X :: $[a,b,c,d]$）のように表現する．記号の場合も制約に対応した値の絞り込みが行われる．X :: $[a,b,c,d]$ を与え，$X \neq a$ なら，X :: $[b,c,d]$ が導かれる．

遅延実行は以下のように記述される．

```
delay predicate(X,...) if <condition>
```

たとえば，delay foo(X) if var(X) なら，X が具体化されるまで foo の実行が遅延される．コルーチンは，これらの機能を利用して記述することができる．

4.2 配置アルゴリズム

我々は，効率化の方法としてコルーチンを利用したアルゴリズムを開発した．以下では，アルゴリズムについて具体的に説明する．

4.2.1 コルーチン処理の方法

ここでは始めに，コルーチン処理を行うことを前提として，制約の処理をメインプロセスで行うものと，サブプロセスで行うものに分けた．

メインプロセスで処理する制約は方位と隣接で，サブプロセスで処理する制約は大きさから考えて，壁に部屋が配置できるかどうかのCapacity制約と，部屋どうしの重なりをチェックするnon-overlap制約，最低限度の面積をチェックする制約である．

面積制約を省略して，処理概念を説明したものが，**図10**である（図中では，上が北方向）．メインプロセスでは，各部屋の配置候補位置と隣接関係によって部屋を配置し，2つのサブプロセスがここでの配置が可能かどうかを監視する．メインプロセスが行った配置を他のプロセスが妥当かどうかチェックする．

たとえば，図10の中央の部屋2が北側（North）という制約に基づいて配置されると（2n）のようにnonoverlapで重ならないことをチェックする．(2c) でcapacityが残りのスペースに部屋が入るかどうかをチェックする．また，図の右端の例では，部屋3を南側（South）にメインプロセスが配置しようとしたときに，(3c) のCapacityが働いて入らないことをチェックし，南側（South）への配置を止めている．

以下では，メインプロセスの処理について説明し，次に3つのプロセスの連携について説明する．

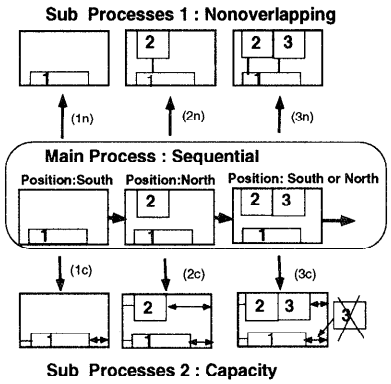


図10 配置のための3つのプロセス
Fig. 10 The three process of layout.

メインプロセスの処理

これは**図11**のアルゴリズムで実現される．始めに，与えられた配置対象の部屋のリストAllRoomsを手続きORDERED-ROOMS（アルゴリズムの(1)）によって並べかえる．ORDERED-ROOMSは，制約の厳しさによって部屋の配置順位を決め[5]，配置順序に対応したRoomListを生成する．制約の厳しさは，隣接の数，部屋の大きさの最小値，あらかじめ設定した優先度（たとえば，玄関は一番最初におく）で決められる．

次に，先の節の領域の知識の追加をAPPEND-KNOWLEDGEによって行い，取り扱う制約の集合ConstraintStoreを生成する（アルゴリズムの(2)）．ここからが，実際の配置処理になる．

また，配置領域に含まれる外壁の情報をGET-WALLSで取り出す（アルゴリズムの(3)）．

この後，RoomListに示される部屋配置の順番に従って配置する部屋がなくなるまで，部屋を順に配置していく（(Loop1)）．PUT-ROOM-BY-POSITIONは，配置位置制約によって部屋を配置する（(4)）．ここでは，方位の制約（東西南北と角のいずれか）によって部屋を配置していくが，もし，配置対象の部屋が隣接関係を持つ部屋があれば（(5)），PUT-ROOM-BY-ADJACENCY（(7)）で配置する．この中では，隣接関係にある部屋を選択した後でその部屋も配置してしまう．

サブプロセスとメインプロセスの連携

アルゴリズムのメインプロセスと，それ以外のサブプロセスの連携は**図12**のアルゴリズムで実現される．なお，具体的な処理と，値の絞り込みについては，次項で示す．

Algorithm PositioningRoom(AllRooms, UserConstraints,KNOWLEDGEBASE)

```
Procedure PositioningRoom (AllRooms, UserConstraints, KNOWLEDGEBASE)
RoomsList ← ORDERED-ROOMS(AllRooms) -(1)
ConstraintsStore ← APPEND-KNOWLEDGE(UserConstraints,KNOWLEDGEBASE)-(2)
PlacedRoomsList ← {}
OuterWalls = GET-WALLS(UserConstraints)-(3)
    while RoomsList is not empty do-(Loop1)
        begin
        R ← SELECT-ROOM(RoomsList)
        Remove R from RoomsList
        PosConstraints ← SELECT-POSITIONING-CONSTRAINTS(R,ConstraintsStore)
        Remove PosConstraints from ConstraintsStore
        PUT-ROOM-BY-POSITION(R,PosConstraints,OuterWalls) -(4)
        AdjConstraints ← SELECT-ADJACENT-CONSTRAINTS(R,Constraints)
        If AdjConstraints = {} then -(5)
            Add R to PlacedRoomsList
        else
            Remove AdjConstrains from ConstraintsStore
            OpositeRooms ←
            SELECT-ROOMS-BY-ADJACENCY(R,ConstraintsStore,AdjConstraits,OuterWalls)-(6)
            PUT-ROOM-BY-ADJACENCY
              (R,OpositeRooms,RoomsList,ConstraintsStore,)  (7)
            Add R to PlacedRoomsList
        end
    end  PlacedRoomsList is a solution

end
end
```

図11 メインプロセスのための配置アルゴリズム
Fig. 11 Layout algorithm for the main process.

(1) PUT-ROOM-BY-POSITION によって部屋が壁に配置された.

(2) 壁配置をトリガとして，以下の制約を同時に適用する.

(2-1) 制約 Capacity を適用する.

(2-2) 制約 non-overlap を適用する.

(3) (2) の処理に応じて以下を実行

(3-1) (2-1)(2-2) のいずれかで失敗した場合
→バックトラックして別の配置候補を探す.

(3-2) (2-1)(2-2) どちらにも失敗しなかった場合
→(2-1)(2-2) の制約が適用された配置の状態で，次の部屋配置に移行.

図12 コルーチンによる配置アルゴリズム
Fig. 12 Layout algorithm based on co-routine.

4.2.2 実　　装

コルーチン処理の実装は，メインプロセスとサブプロセス間に制約論理型言語の領域を持つ共有変数を導入して，遅延実行機能を使った共有変数の挙動の変化によって，お互いのプロセスに変化を起こさせることで行っている.

具体的には以下のとおりである.

始めに，配置位置を表す共有変数 W_i（i は部屋の ID）を導入する．W_i は，

$W_i :: wall_1, ..., wall_j, ..., corner_1, ..., corner_k, ...$

であり，$wall_j$ は，配置領域に含まれる外壁の ID で，配置領域が長方形なら最後は 4 である．$corner_k$ は，配置領域に含まれる角の ID で，配置領域が長方形なら最後は同じく 4 である.

メインプロセスで部屋が配置されると，以下のルールに従って，共有変数の値を具体化（Wi の値を決めること）し，サブプロセス 1，2 を起動する（説明を分かりやすくするために，角への配置の場合の説明は省略する）.

If $Room_i$ が $wall_j$ に配置 **then** $W_i = wall_j$.

サブプロセス 1 においては，述語 delay を利用して W_i の値が具体化するまで実行が遅延されていた non-overlap 制約が起動される．新しくおかれた部屋 i とすでに配置されている部屋との間に制約が適用され，制約が満足されなければ，non-overlap を失敗させることで，$W_i \neq wall_j$ をメインプロセスに伝える.

サブプロセス 2 の Capacity も起動されるが，サブプロセス 1 の場合より少し複雑である.

始めに Capacity 制約の定義は，領域を $\{0,1\}$ とする変数 B_{ij}（i は部屋の ID，部屋の数は n 個とする）を使って，以下のようになる（L_i は，$Room_i$ の幅か奥行き（$wall_j$ の向きによる））.

Capacity 制約：

$$B_{1j} * L_1 + ... + ... + B_{ij} * Li + ... + B_{nj} * L_n \leq \text{LengthOf}wall_j(wall_j\text{の長さ})$$

Capacity の場合，W_i の具体化の影響が，W_i と変数 B_{ij} の以下の関係によって伝播する.

If $W_i = wall_j$ **then** $B_{ij} = 1$. ——(1)

If $B_{ij} = 0$ **then** $W_i \neq wall_j$

($Room_n$ は $OuterWall_m$ に配置できない)
——(2)

メインプロセスからサブプロセスへの伝播が(1)の関係によって行われ，逆の関係による伝播が(2)で起こる．後者のケースで $B_{ij}=0$ が導かれる理由は，(1)による制約伝播で，B_{ij} が順次具体化していく際，上記の Capacity 制約の定義に基づく計算が行われるためである．

たとえば，配置対象の部屋が3つ（部屋1 10×10，部屋2 20×20，部屋3 30×30（部屋名サイズ（縦×横））で，このうち，部屋1，部屋2が長さ40の同じ $wall_j$ に隣接して配置された場合，Capacity 制約は，

$$1*10+1*20+B_{3j}*30 \leq 40$$

になる．この結果，代数計算が行われ，$B_{3j}=0$ が導かれ，部屋3は $wall_j$ には配置できないことが導かれる．

このように，サブプロセス2の Capacity によって，メインプロセスで未配置の候補の配置可能性を絞り込むことができる．

4.3 値の決定

抽象的な配置のままでは図面として表示することはできないので，値を決める必要がある．このためには，いくつかの方法があるが，ここでは，すべての部屋の大きさの合計を最大にするために，すき間を最小にするという方法をとる（**図13** の例では，右側の図の(1)のすき間を最小にすること）．

Capacity 制約にすき間に相当する Space という変数を加え，$minimize(Space)$ を計算させることで実現する．

$$\sum_{j=1}^{n} l_i + Space = \mathrm{LengthOf}wall_j$$

l_i は，同じ外壁 $Wall_j$ を共有する部屋の奥行か幅を表す．

外壁沿いの長さを決定した後，壁に接しない側の部屋の長さを決定する必要がある．これは，長さが最大になるように決定していく．図13では，左側の図の矢印(2)で示す部分を最大にすることである．いずれの値の決定の場合も，エスキースの作成の場面で空間的な制約は満足しているので，バックトラックは発生しない．

5. 応用例と評価

本章では，我々の方法の有効性を示すために，前章までの方法で作成した住宅間取作成システム TG-FP

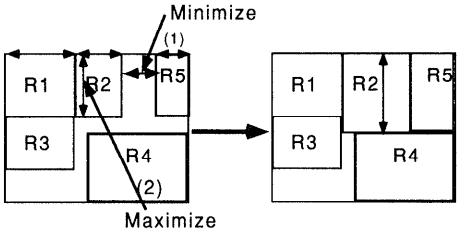


図13 値の決定の例
Fig. 13 Instatiation of variable.

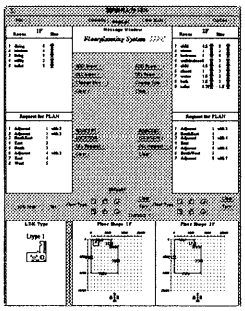


図14 入力パネル
Fig. 14 Input panel.

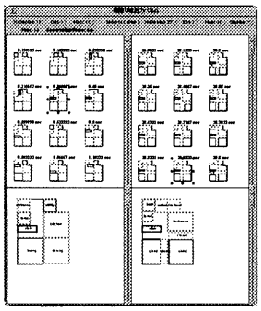


図15 出力パネル
Fig. 15 Output panel.

（Tokyo Gas Floor Planner）を紹介する．

5.1 TG-FP

我々は，先のアルゴリズムを基に住宅間取作成システム TG-FP（Tokyo Gas Floor Planner）を実現した．

TG-FP は，他のフロアプランニングの自動化システムと同様に，設計を依頼するクライアントの要望を明確化するための設計の専門家が作成する第一次案の設計支援がターゲットである．最終的な詳細設計を行うのは，あくまでも専門家という立場のシステムである．このために，複数の定性的な位置関係の出力と，出力の修正機能を持たせている．

以下では，実際の例を使って TG-FP を説明する．

TG-FP は，**図14** の入力パネルを使って設計に必要な条件を入力する．最上部の2つのパネルで，それぞれ1階と2階に必要な部屋を入力する．その下の2つのパネルはそれぞれの階の部屋に関する制約条件を入力する．最下部の2つのパネルで，1階と2階の配置対象領域と玄関の位置を入力する．左下のパネルでは，LDK の配置をパターンで選ぶこともできる．入力後，**図15** のウィンドウで MakePlan のボタンを押すと，1階のレイアウト作成が実行される．レイアウトは複数案表示でき，ユーザはこの中から好みのものを選択することができる．選択したものに対して，2階を作成するには，MakeFloorPlan 2F のボタンを押

表1 TG-FPとLIFEによる実行性能の比較
Table 1 Performance: two types of TG-FP.

ケース	部屋数	LIFE（コルーチンなし）			TG-FP（コルーチン処理あり）		
		実行時間（秒）	実行ステップ数	バックトラック数	実行時間（秒）	実行ステップ数	バックトラック数
1	11	1.38	67,904	19,743	0.73	56,646	17,228
2	10	24.28	591,666	98,252	0.73	21,022	3,679
3	10	20.65	669,202	117,429	1.18	25,715	5,187
4	11	–	–	–	81.6	1,418,654	468,810
5	10	–	–	–	6.92	248,446	80,167
6	10	20.11	857,411	152,746	5.42	203,211	65,979
7	10	–	–	–	6.70	221,426	63,895
8	10	68.27	1,996,377	464,607	1.72	56,646	17,228
9	9	30.55	10,848,146	1,859,337	4.32	158,991	46,245
10	10	–	–	–	0.67	23441	6657

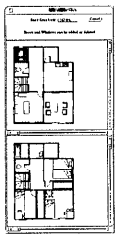


図16 簡易CADによる修正例
Fig. 16 The output by using small-CAD.

せばよい．

また，出力結果を付属の簡易CADを利用して修正変更できる．一例を図16に示す．

5.2 評　価

本節では，我々のアプローチを，システムであるTG-FPの実現の観点と，配置アルゴリズムとその実現方法の観点から評価を行う．

5.2.1 自動レイアウトシステムとしての評価

表1の右側は，TG-FPによる実行結果である[8]．実行時間と，実行ステップ数，バックトラック数を示してある．

また，コルーチン処理を利用したTG-FPの有効性を測定するために，左側に我々が以前に作成したコルーチン処理を用いていないシステムLIFEの実行結果も示してある[7]．

表中の「–」は，10分以内に答を得られなかったケースであり，実行を打ち切っている．

システムを実際に使うという観点から見ると，実行時間と，解を得られる問題の数，出力される答の妥当性が重要になる．表を見ると，実行時間は，TG-FPはおおむね1分以内で解を得ており，従来のLIFEに比べて，より多くの問題を解いている．また，ここで得られた答を，数人の専門家に見せているが，(1)複数のレイアウトを考えるうえでのヒントになる，(2)お客に複数の図面を見せ，好きなものを選んでもらうことで好みを調べるのに有効，という評価を得ており，当初の第一次案の設計支援の目的を満足している．

5.2.2 アルゴリズムの評価

ここでは，コルーチンベースの我々のアルゴリズムの評価を行う．TG-FPの実現を通して，我々のアルゴリズムでは，以下が実現できた．

(1) Generate & Test型の処理において，Testをより完全に行えた．
(2) 制約伝播を効率的に利用した．
(3) 制約の宣言的な記述を実現した．
(4) 少ないコード量で実現できた．

non-overlap制約の適用などで実現されている(1)や，capacity制約の適用などで実現されている(2)は，システムを実際に利用するうえで重要な，実用的な時間で答を得るために必要な特徴である．実際にどれくらい(1)，(2)が有効であったかを表1を使って説明する．

表1において，コルーチンを使っていないLIFEのものより一般的に早く，総実行ステップ数で見ても，少ない回数で答を得ている．効率の観点からは，表1の探索の際のやり直しの回数であるバックトラック数に着目することができる．たとえば，ケース8の場合は，使わなかった場合に464,607回だったものが，コルーチン処理を使ったものは，わずか17,228回数になっている．他のケース2，3，9などの例でも分かるように，実行時間，実行ステップ数の差が大きいほど，バックトラック数の差が大きい．これらの結果からは，コルーチンを使ったものの方が，制約伝播によって，値の絞り込みがより頻繁に起こり，さらに，答に至らない候補を，早い配置レベルで枝刈りしていることが分かる．

(3)，(4)は，作成したシステムを，様々な環境変化（システムの維持管理者の変更，対象とする問題の追加）に対応して継続的に利用するうえで重要である．

空間配置制約を定義し，これを利用して，フロアプランニング問題を宣言的に記述している（(3)）ので，ルール等で問題を記述する場合に比べて，プログラムが読みやすい．また，実際のプログラムコードの量の面で見ても，制約処理を行う部分を500行程度で実現（(4)）しており，メンテナンス性や，拡張性の向上の点でのメリットが期待できる．

6. おわりに

本稿では，制約に基づくフロアプランニング自動化システムの実現方法について述べた．

フロアプランニング問題を制約で表現し，制約を効率的に処理する方法を導入することで問題を解く方法を示した．具体的には方程式と論理式からなる空間配置制約を定義し，これらを使ってフロアプランニング問題を定式化した．さらに，空間配置制約を処理して解を得るための方法として，組合せの削減を実現するコルーチン機能に基づくアルゴリズムを提案した．

一方，我々の方法を，住宅間取自動作成システムTG-FPに適用し，TG-FPの実用性の観点から，実行時間と出力結果が妥当であることを確認した．また，実現したアルゴリズムは，効率的な探索を実現しており，しかも，制約論理型言語を使って，短い行数で，しかも拡張性のある形でのコーディングができることを確認した．

今後は，本手法をより幅広いフロアプランニング問題に適用するとともに，専門家の設計支援の立場から，システムの出力を専門家がより容易に修正，変更していける知的CAD機能の追加を検討していく．

謝辞 日頃，自動レイアウト作成システムの実現および論文作成に関して様々なご指導をいただいている東京理科大学理工学部の溝口文雄教授，大和田勇人講師に謹しんで感謝いたします．また，システム実現にあたり協力していただいた東京ガスハウジング（株）の関係者の方々に感謝いたします．あわせて，丁寧な査読をしていただいた査読者の方々に感謝いたします．

参考文献

1) Baycan, C.: Formulating Spatial Layout as a Disjunctive Constraint Satisfaction Problem, PhD Thesis, Carnegie Mellon University (1991).
2) Cohen, J.: Constraint Logic Programming Languages, *Comm. ACM*, Vol.33, No.1, pp.52–68 (1992).
3) Enomoto, H., et al., Development of Office Planning Expert System, *Proc. 4th Conference on Computing in Civil and Building Engineering*, p.248 (1991).
4) Flemming, U.: Rule-Based System in Computer-Aided Architectural Design, *Expert Systems for Engineering Design*, pp.93–112, Academic Press, London (1988).
5) Fox, M., Zadeh, N. and Baykan, C.: Constrained heuristic search, *Proc. IJCAI-89*, pp.309–315 (1989).
6) Hashimshony, R. and Shaviv, E.: Transforming an Adjacency Matrix into a Planar Graph, *Building and Environment*, Vol.15, pp.205–217 (1980).
7) Honda, K., et al.: A Floorplanning System Using Constraint Logic Programming, *Proc. 2nd Practical Applications of Prolog* (1994).
8) Honda, K. and Mizoguchi, F.: Constraint-based approach for spatial layout planning, *Proc. of 11th Conference on Artificial Intelligence for Applications*, pp.38–45, IEEE Computer Scociety (1995).
9) Jampel, M.: Over-Constrained Systems in CLP and CSP, PhD Thesis, City University (Sept. 1996).
10) 川窪広明，辻 正矩：隣接グラフから外面条件を考慮した平面グラフを求める方法について，第17回情報システム利用技術シンポジウム，pp.151–156 (1994).
11) Maher, M.: ECLIPSE 3.5 USERS MANUAL, ECRC (1995).
12) 大野，大岡：探索アルゴリズムを用いた住宅間取り自動作成システム，第13回情報システム利用技術シンポジウム (1990).
13) 岩井一幸，奥田宗幸：すまいの寸法，計画事典，彰国社 (1992).
14) Pfefferkorn, C.: A heuristic problem solving design system for equipment or furniture layouts, *Comm. ACM*, Vol.18, No.5, pp.286–297 (1975).

(平成9年1月14日受付)
(平成9年9月10日採録)

本多 一賀（正会員）

昭和38年生．昭和63年東京理科大学理工学研究科経営工学専攻修士課程修了．同年東京ガス（株）入社．現在情報通信部IT応用開発センター主任研究員．AIおよびインターネット関連技術を活用したインテリジェントシステムの研究開発に従事．ACM，AAAI会員．