倉庫番手詰り判定手法の計算効率の検証

小田原大　川合慧
{haradai,kawai}@graco.c.u-tokyo.ac.jp
東京大学大学院総合文化研究科広域システム科学系

概要

文献 [1] で提案した倉庫番における手詰り判定手法に改良を加えるとともに、その計算効率を実験によって検証し、確認した。コンピュータパズルの倉庫番のソルバにおける課題は、手詰りの局面を効率良く検出することである。特に袋小路の判定は難しい問題とみなされている。

ここで扱う手詰り判定アルゴリズムの理論的な計算量はユニットの個数に関し指数関数的に増加するため、実用上の有効性が問われる。本稿では袋小路を含めた「手詰り」状態の一般的な判定方法を実際の問題に適用し、一般に難しいとされる問題において実際に必要な計算量を求め、提案する手詰り判定手法の有効性を示した。

Practical Computational Cost for Detecting Deadlocks in Sokoban Puzzle

Masaru Odawara　Satoru Kawai

Department of General Systems Studies,
Graduate School of Arts and Sciences, University

The computational cost of an improved version of formerly proposed method for the deadlock detection in Sokoban puzzle is calculated for some practical configulations of the puzzle. Though, theoretically, the size of the search space for the method increases exponentially, the practical behaviour of the method is proved to be sufficiently tractable, showing the effectiveness of the method.

1　はじめに

コンピュータパズル倉庫番のソルバーの開発において、手詰りの判定の困難さが指摘されている。本稿では、文献 [1] で提案された倉庫番における手詰り判定手法を改良するとともに、適用の際の計算効率を実験によって検証した。

図 1: 地形: 左から順に番人、荷物、ゴール、スペース、壁、ゴール上の荷物

1.1　倉庫番用語

本稿では倉庫番を構成する要素を図 1 のように呼び表すこととする。

これらの要素によって構成されるものをマップと

呼び、マップの一部分を取り出したものを部分マップと呼ぶ。また、マップによって表される問題を面と呼ぶ。手詰りとは、下に示す「死に手」もしくは「不能状態」に陥った状況を指す。この状態からその面 (倉庫番の問題の単位) をクリアすることはできない。

死に手　ゴール上にない荷物を一マスも動かせなくなった状態。

不能状態　死に手荷物を押すことは可能だが、決してゴールに運べない荷物を含むような状態。

1.2 関連研究

手詰りの判定プロセスについては上野 [3] で触れられている。ここでは部分的な不能状態を判定するために、直接不能状態を構成していない荷物を除き、手詰り判定のルーチンを起動させていた。しかし荷物を除くことで手詰り判定が失敗したり、ルーチン内での荷物移動に伴いルーチンの再帰呼び出しが必要になる場合が生じるといった問題点が存在した。

また、マップ全体の荷物の配置に依存して形成されるような不能状態 (ここでは全局的な不能状態という) に対する判定はもとより対象とならなかった。

Andreas[6] は、ソルバー Rolling Stone において手詰り判定としてはローカルなデータベースを探索拡張の一手段として用いた。この手法は全局的な手詰りには対応していない。

これらの問題点に対して、小田原 [1] は全局的な手詰りの判定手法を提案したが、この手法で用いられたユニットの概念は倉庫番の問題のモデル化としては不十分な点が多く、現実の問題に適用することは難しい場合があった。

本稿では、手詰り判定手法 [1] をベースにそのユニット概念を改良して計算量の見積もりと実験を行なった。以下、2 章では定義の修正を行ない、3 章で計算量の理論的な見積もりをたて、4 章では実験の条件とその結果を述べる。5 章でまとめと今後の課題を述べる。

2 手法の修正

文献 [1] におけるユニットは小さなユニットにおいてのみ成り立つような制限が設定されていた。また、ユニットとユニット間通路とを異なる概念とみなしたため、適用が難しい場合があった。

本稿では実際に倉庫番のマップに適用して実験するに当たり、ユニットの概念を再定義し、上の制約を減らし、より一般的に適用可能なものとした。

2.1 ユニットの定義

ユニットとは、部分マップを表している概念であり、出入口と内部状態をもつ。倉庫番のマップは部分マップの集合として表され、部分マップは全てユニットとして表現することができる。マップはユニットの組合せとして表現することができる。

ユニットは出入口を備えており、番人は同一ユニットのある出入口からある出入口へと移動することでユニットを通過することができる。

出入口の組をパスといい、通過可能なものと通過不可能なものがある。ユニットの内部状態は、通過可能なパスの集合で表す。

内部状態の変化　番人があるユニットの 1 つのパスを通過すると、ユニット内の荷物の配置が変化するため、可能パスが変化 (=内部状態が変化) する。番人が同一のパスを通過する場合であっても、結果としてより多くのパスが可能パスとなるようにユニットの内部を移動することができなる場合がある。この点を考慮に入れ、手詰り判定の手法に以下の改良を加える。

2.2 定義の変化に伴う変更点

文献 [1] と、本稿における概念/手法の扱いの違いを示す。

- 手詰り判定の方法
- パス通過の際のユニット内部における探索
- 静的なパスと動的なパス
- パスの半順序集合

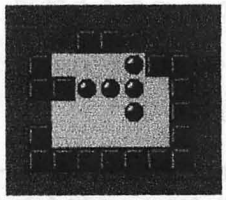

図 2: xsokoban[8] level 2 の部分マップ

- 荷物の押し出しと運び込み
- グローバル探索の手法
- 番人の最初の移動

手詰り判定の方法　従来は全ての「ユニット間通路」に到達可能であるかを手詰りの判定基準としていたが、本稿では「ユニット間通路」という概念を排除したため、全てのユニットの出入口に到達可能かどうか、を手詰り判定の基準と見なす。

パス通過の際のユニット内部における探索　新たに内部における探索を行なうことにする。出入口 a から出入口 b に番人が抜けるパスを考えた時、単純に番人がパスを一直線に通過するだけではなく、荷物の移動距離を定め、その移動距離の中で全ての荷物の配置を調べる。ただし、全てのパスが可能パスとなった時点で探索を中止する。この際、内部の可能パスが変化するが、その変化は単純に一通りではなく複数通りになりうる。

この際、荷物の移動に伴いユニット内部で手詰りが生じることがある。これについては、ローカルに手詰り判定を行ない、四つ組、"シチョウ"[1]などの死に手を形状をパターンとして手詰りを未然に防ぐ。

例えば出入口 a, b, c を備えたユニットに相当する部分マップ (図 2) が存在した場合、パスは (a,a),(a,b),(a,c),(b,a),(b,b),(b,c),(c, a),(c, b),(c,c) の 9 通りが存在する。これらのパスを番人が実際に通過させ、通過可能であるようなパスを選ぶ。これが最初に得られる可能パスの集合である。ここでは (c, b) の 1 通り である。ここで、各可能パスについてローカル探索を行ない、この可能パスを通過した際に生じる静的な可能パス集合から、極大のパスの集合を求める。

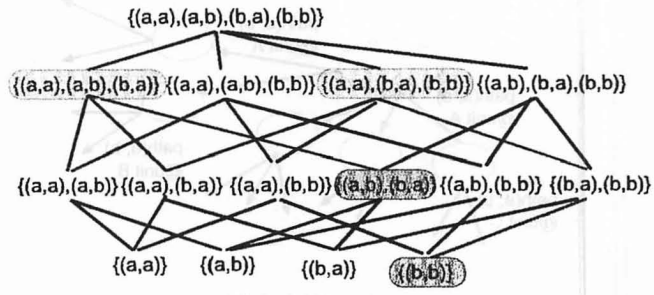

図 3: 状態の束

これが状態遷移後のユニットの状態である。1 つの可能パスを通過すると複数の異なるパス集合が生じる可能性がある。つまり 1 つの可能パスの通過によって複数の状態へ遷移することがある。

静的なパスと動的なパス　ただし、この可能パスを通過した後に得られる可能パスは当初設定している可能パスとは意味が異なる。ユニットの初期状態において最初に求めた可能パスは、通過によって変化するのに対し、状態変化によって得られる可能パスは、そのパスを通過してもユニットの状態を変化させない。前者の可能パスを動的なパスといい、後者を静的なパスという。

ユニットの状態変化は一度のみとする。一度ユニットの状態が変化した後には、状態変化は生じないことにする。

ユニットの状態が {(a,a)} であったところで、(a,a) を通過することによって状態が {(a,a), (a,b)} に変化したとする。しかしこの状態から (a,a)(a,b) のパスを通過しても、内部の状態変化は生じない。

パスの半順序集合　可能パスが複数通り存在した場合、図 3 のようにパスの集合を包含関係によって半順序を定めることができる。

あるパスを通過した時に可能パスが複数得られた場合、あるパス p_0 がパス p_1 を包含する場合、p_1 のパスを可能パスに加える必要はないとみなして p_0 のみを採用する。

[1]文献 [3]

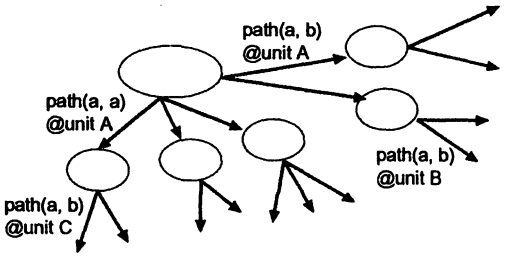

図 4: 探索空間

荷物の押し出し、運び込みに関して　従来は荷物の押し出しを無視して考えていたが、本稿での改善手法においては、荷物を運び出した場合、隣接したユニットの出入口へと荷物を運び込むことにする。その際、隣接したユニットの内部状態は考慮しないが、出入口に新しい荷物を配置する。

グローバル探索　グローバルに探索を行なう手順は、状態が複数に遷移するこの場合においては文献 [1] とは異なる。

探索の目的は、番人が全てのユニット境界に到達するような局面を発見することである。探索空間は図 4 のように、通過するユニットの状態によって分岐が生じる局面の集合である。この局面の集合をユニットの状態遷移を考慮しながら探索する。

最初の一手　番人は従来はユニット間通路に存在するとしていたが、本稿においては番人は常にユニット内に存在している。したがって番人が別のユニットに移動するためにはユニット内をスタート地点とし、その出入口へ至る過程でのユニットの状態の変化を反映させる必要がある。

最初の番人の移動については、現在番人がいるユニット内の場所から各出入口に至るまでに移動した荷物の位置をユニットの初期状態に反映させることとする。

2.2.1 探索の手順

- ユニットに分割する
- ユニットの内部状態を調べる (パスを通過させてみる。このときには荷物の移動を含む。)
  →可能パス (a) の集合を確認
- この際、パス間の包含関係による可能パスの選別は行なわない。
- 可能パス (a) に基づきユニット内の探索を行ない、可能パス (b) の集合を求める
- 可能パス (b) の集合に対し、包含関係による可能パスの選別を行なう。
- ユニットの各状態と遷移先の状態が定まったことにより、グローバル探索の準備が整う。
- グローバル探索を行なう。ユニットのあらゆる状態について全て局面を生成し、全ユニット境界に到達可能な局面を求める (※)。

可能パス (a) は動的なパス、可能パス (b) は静的なパスを意味している。

探索の起点は、番人が存在するユニットである。上で述べたように、番人の存在する地点から、そのユニットの出入口へ番人が移動し、そこから探索を開始する。

(※) 番人が全ユニット境界に到達可能である時には手詰りではない。そうでない時には手詰りである可能性がある。

3 計算量

ユニット内部の探索　ユニット内部を番人が動き回り局面を探索することで、可能パスが最も大きくなる局面を求める。ここでは局面を全探索するが、全てのパスが可能パスと判断される局面が生じた時点で探索を打ち切る。

ユニットがとりうる状態の個数　ユニットにおける出入口の数を n 個とすると可能パスの個数は n^2 個となる。このとき、可能パスの組合せの個数は全て可能な場合からどれも不可能である場合を含めて 2^{n^2} 通り。半順序を定め、包含関係によって 極大となるパスのみを選ぶとしても最大のパスの個数は 約 ${}_{n^2}C_{(n^2/2)}$ 通りである (n が 4 の時、12,870 通り)。つまり、可能パスの組合せの個数 (= ユニットの状態の個数) は出入口の数に対して指数関数的に増加する。

ユニットを組み合わせた時のグローバル探索　番人のユニット間の移動に伴い生成される局面の数は、番人が通過するユニットの個数に関して指数関数的に増加する。

単純な例を図5に示すと、ユニットがU_A, U_B, U_Cの3つ存在するとする。最初番人はU_Aのパス(b, a)を通過する。その際、U_Aを通過することでU_Aの状態は変化し、固定される。ここで、U_Bの状態が(a, a)のみ可能パスだったとする。番人は(a, a)のパスを通過すると、番人の位置には変わりはないが、U_Bの状態が{(a, b), (a, a)}になったとする(図5の3番目)。ここで番人は(a,b)を通過してU_Cへ移動することができる。ここでU_Bを通過したため、U_Bの状態は変化し、固定状態になる。

このようにユニットの状態を変えつつユニットを通過し、最終的に全ユニット間を渡ることができる局面に到達できればグローバル探索は終了する。

この手法にしたがい「手詰りではない」と判定するためには、番人がマップを構成するユニットを全て通過しなければならない。ユニットの個数をn個とし、各ユニットの持つ出入口の平均個数をk個とする。さらに各ユニットの状態の遷移先の平均個数をs個とする。荷物の押し出し、運び込みを考慮すると、1つの状態に対し、2^k個の状態を考慮する必要があるため、ユニットを全て通過するまでに生じる局面数は$(2^k \cdot s)^n$に比例する。

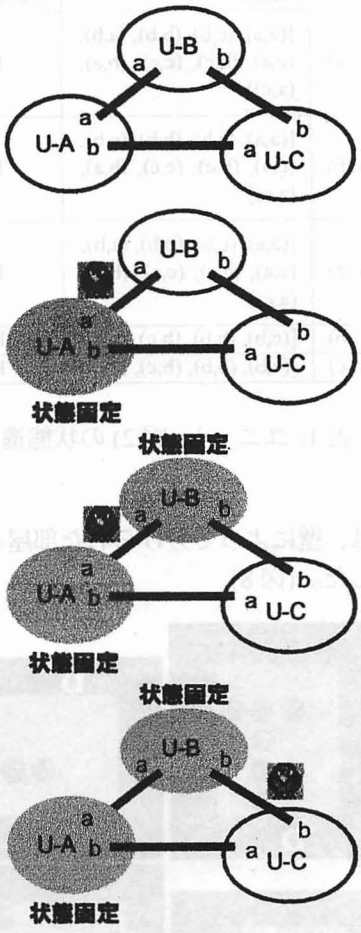

図5: グローバル探索の例

4　実験

本稿では上の前提にしたがい、倉庫番の実際の問題セットの一部をユニットに分割し、出入口を設定して

1. ローカルなユニット内部の探索(ローカル探索)に基づく、状態遷移先の数
2. 問題を構成する複数のユニットについての探索(グローバル探索)

について調べる。

4.1　対象とするマップ

ゴールエリア[2]とその他のマップが分かれたもの、また壁によって部屋が区切られ、ユニットに分割しやすいようなマップを主に対象とした。

対象の問題はgsokoban[9]に含まれている問題セットのうち、**Sasquatch**を主に用いた。また、xsokoban[8]の問題も用いた。これらの一部をユニットに分割して実験を行なった。

4.2　ユニットの分割手法

荷物の押し出し、運び込みが実装できれば、ユニットの分割位置にはそれほど関わらないと思われるマップを選んだため、ユニットの分割手法に

[2]文献[2]

path	status	# of status
(a, a)	[(a,a), (c,b), (b,b), (a,b), (c,a), (b,c), (c,c), (b,a), (a,c)]	1
(a, b)	[(a,a), (c,b), (b,b), (a,b), (c,a), (b,c), (c,c), (b,a), (a,c)]	1
(a, c)	[(a,a), (c,b), (b,b), (a,b), (c,a), (b,c), (c,c), (b,a), (a,c)]	1
(c, b)	[(c,b), (b,b), (b,c), (c,c)]	1
(c, c)	[(c,b), (b,b), (b,c), (c,c)]	1

表 1: ユニット (図 2) の状態遷移

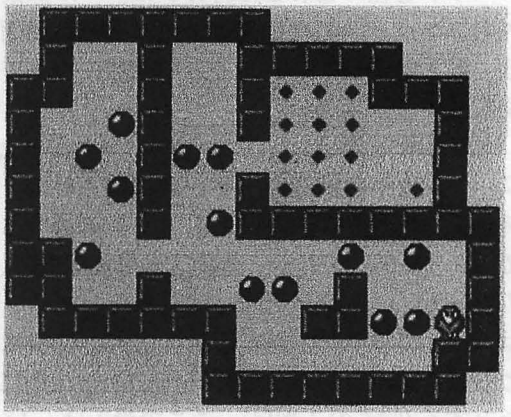

図 7: 問題マップ

ついては、壁によって分けられた部屋ごとに手動で分割した。(図 8)

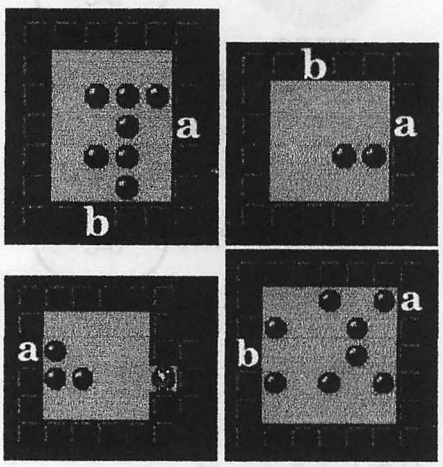

図 6: xsokoban level4 のユニット

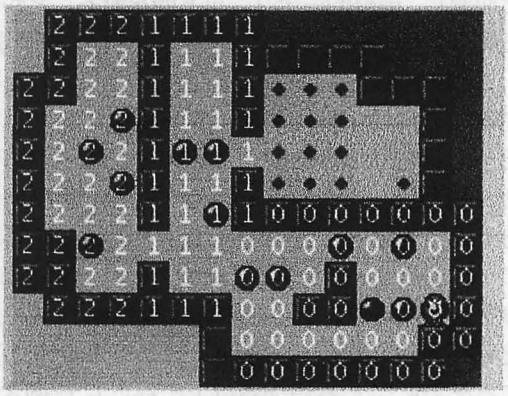

図 8: 問題マップのユニットへの分割 (数字とユニットが対応)

4.3 ローカル探索

分割した各ユニットに対し、ローカル探索を行なった。ユニットのローカル探索の結果を下表に掲載する

表 1、2 は、xsokoban の level.2、xsokoban level.4 の問題図の部分マップ (図 2, 6) を出入口 a, b, c を持つユニットとしたときに、全てのパスに関して通過した際に遷移する状態を表したものである。

この結果によると、a から入って a, b, c から出るパスについては、どれも全てのパスを可能パス

path	status	# of status
(a, a)	[(a,a), (b,b), (a,b), (b,a)]	1
(a, b)	[(a,a), (b,b), (a,b), (b,a)]	1
(b, a)	[(a,a), (b,b), (a,b), (b,a)]	1
(b, b)	[(a,a), (b,b), (a,b), (b,a)]	1

表 2: ユニット (図 6) の状態遷移

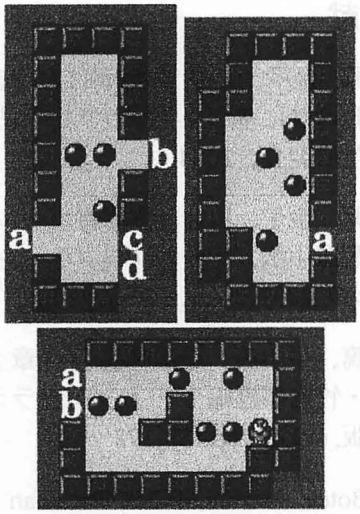

図 9: 問題マップのユニットとなる部分マップ。左上:ユニット 1、右上: ユニット 2、下:ユニット 0

とするような経路が存在しており、結果として状態の遷移先は 1 通りに定まる。

s は実験した問題に関して全て 1 となった。状態遷移の個数は計算量において現実的には考慮する必要がないと考えられる。

4.4 グローバル探索

ユニットを組み合わせた際のグローバル探索の実験を行なう。対象としたのは上で用いたユニットからなるマップである。

計算量に関わってくる要素は上に述べたように

- ユニットの個数
- 出入口の数
- 状態遷移先の個数

の 3 つである。表 3 に、各マップを構成するユニットについて上の 3 つの要素を記した。なお Sas、Xsoko はそれぞれ問題セット Sasquatch、Xsokoban を意味する。

例えば図 7 のマップは、xsokoban level26 の問題の初期状態を表している。初期状態であるから当然手詰りではない。図 8 のように手動でユニットに分割 (ユニットの図 9) した上でローカル探索を行なった結果以下のように状態が定められた。

問題	n	s	k
Xsoko 4	4	1	1.75
Xsoko 26	3	1	2.33
Sas 10	5	1	3
Sas 12	5	1	2.6
Sas 13	4	1	2
Sas 15	3	1	2
Sas 19	3	1	3.67
Sas 22	3	1	2

表 3: 各問題における n, s, k の値

- ユニット 0 : b を入口とするもの以外全てのパスを初期状態とし、通過後には全パスが可能パスとなる
- ユニット 1 : b を入口とするもの以外全てのパスを初期状態とし、通過後には全パスが可能パスとなる
- ユニット 2 : パスは (a, a) のみで、通過後の可能パスも同様である。

なお、図 10 の左上のユニットの b にはゴールエリアのみのユニットが接続しているためにここでは省いた。上の状態は、番人がユニットの外側にいる時を前提にしている。これらの状態を用いて、現在の手詰り判定を行なう。

番人はユニット 0 から出る時に、a または b から出る。この時状態が分岐するが、**b** から出た場合のうち、a から荷物を押し出した上で b から出るという手が選択される必要がある…(*)。それ以外の場合は直ちにローカルに手詰りが導かれ、ユニット 0 に接するユニット 1 の状態を考慮した段階でこの分岐については探索を終了する。**a** から出た場合には、荷物をユニット 1 の出入口 c から運び込みながら番人が a から出る…(**) ことになる。この場合 (*) にしても (**) にしても、ユニット 2 への境界、ゴールエリアへの境界へは (ローカルの手詰りを回避しつつ) 直ちに到達することができるため、グローバル探索はこの 2 手のいずれかを探索した段階で終了する。

4.5 結果と考察

ローカル探索の結果から、実際にはグローバル探索に費やす時間は s の値を考慮する必要はなく、$(2^k)^n$ に近い値に抑えられると考えられる。実験で用いた問題では平均ユニット個数・出入口の平均個数 (n, k) はどちらも 5 を超えることはなかった。2^k は通常の問題においては指数関数的増加を気にすることなく扱うことができる。特にユニットの個数は小さい値で抑えられるため、グローバル探索に費やす計算量はローカル探索に対して現実的な時間で扱えると考えられる。

5 まとめと今後の課題

本稿では文献 [1] で提案された倉庫番における手詰り判定手法に改良を加えるとともに、適用の際の計算効率を実験によって検証した。結果として多くの問題においてユニット内の状態遷移については状態遷移の個数は 1 に近い値に抑えられ、本手法による手詰り判定は理論的には非常に大きな計算量を要するとはいえ、実際的な問題に対しては小さいコストで判定できる。

課題としては、ユニットの分割手法が挙げられる。本手法でのユニットは、出入口がはっきりした部屋の形になっている部分マップをユニットとみなすことがより有効と考えられるが、そのように部分マップを区切った場合にはユニットのサイズが大きくなり、ローカル探索の負荷が大きくなりすぎる場合がある。本稿では扱っていなかったが、手法を適用するタイミングについても考慮しなければならない。上野ら [3] による閉領域の出現に伴う判定は一つの選択肢である。また、本稿ではユニットの状態変化が 1 度きりとしたが、これについても複数回の状態変化を許容するモデルを検討したい。

謝辞

アドバイスを頂いた KY ゼミ、GPS の皆様に感謝します。

参考文献

[1] 小田原、金子、川合、倉庫番における部分マップの組合せに基づく手詰り判定手法、ゲーム情報学研究会、2004

[2] 小田原、金子、川合、倉庫番ゴールエリアの自動配置手法、ゲームプログラミングワークショップ 2003、Nov.,2003

[3] 上野篤、中山康、疋田輝雄 第 3 章 倉庫番 松原仁・竹内郁雄編 ゲームプログラミング, 共立出版, (1998), pp.158-172

[4] Adi Botea, Martin Müller, Jonathan Schaeffer, Using Abstraction for Planning in Sokoban, CG 2002, LNCS 2883, pp. 360-375, 2003

[5] Andreas Junghanns, Jonathan Schaeffer, Single-Agent Search in the Presence of Deadlocks, Proceedings of the fifteenth national/tenth conference on Artificial intelligence/Innovative applications of artificial intelligence, (1998), pp.419-424

[6] Andreas Junghanns, Pushing the Limits: New Developments in Single-Agent Search, Department of Computing Science University of Alberta Ph.D.thesis, 1999.

[7] "Rolling Stone", http://www.cs.ualberta.ca/%7Egames/Sokoban/program.html

[8] Xsokoban, http://www.cs.cornell.edu/andru/xsokoban.html

[9] gsokoban, http://home.swipnet.se/darshiva/gsokoban/