探索アルゴリズムの並列化とプランニングへの応用

Parallel Search and Its Application to Planning

岸本 章宏[*1*2] Alex Fukunaga[*3] Adi Botea[*4]
Akihiro Kishimoto

[*1]東京工業大学大学院情報理工学研究科
Graduate School of Information Science and Engineering, Tokyo Institute of Technology

[*2]科学技術振興機構 さきがけ
JST PRESTO

[*3]東京工業大学グローバルエッジ研究院
Global Edge Institute, Tokyo Institute of Technology

[*4]NICTA

Developing an efficient algorithm for planning is a classical problem in AI. Current state-of-the-art optimal planners are based on heuristic search with an automatically generated heuristic functions. This paper presents a parallel search algorithm applied to optimal planning. Experimental results show that the parallelized algorithm achieves reasonable speedups and increases the range of problems that can be solved by an optimal planner.

1. はじめに

人工知能の代表的な研究分野であるプランニングの研究では，ユーザーの記述した問題と目標に対するプランを高速にかつ自動的に返すことが目標である．古典的プランニングは，状態，状態を遷移させる演算子であるオペレーター，初期状態，および目標状態から構成される．これらは，有限の記号列で表現できる．本論文のプランニングでは，初期状態から目標状態に到達できるオペレータ列の中で，列の長さが最小の解 (最適解) の発見が目的である．この問題は，状態を探索木の節点，オペレータを探索木の枝に対応させれば，探索問題に帰着できるので，高性能なプランニング・システム (プランナー) の多くは，プラン発見のために探索アルゴリズムを利用している．

探索に利用するヒューリスティックの自動抽出技術の進歩により，プランナーの性能は著しく向上した [Helmert 07] が，最適解を保証する場合の難しさには，プランナーが調べる探索空間の大きさがある．プランニングの探索空間は，組み合わせ爆発が生じるので，なるべく正確なヒューリスティックを抽出して，探索空間を削減したい．しかし，正確でかつ最適性を保証する条件を満たすヒューリスティック関数を自動的に作ることは難しいので，求解のためにプランナーが調べる探索空間は莫大になる．

最近の計算機環境では，内部に複数の CPU コアを持つ計算機だけでなく，ネットワークでつながれていた計算機が普及しているので，これらの環境を利用して探索アルゴリズムを並列化することは，自然な考え方である．

本論文では，プランナーの探索アルゴリズムの並列化を行う．この方法では，最適解が保証できる上に，ネットワーク通信が必要な PC クラスター上でも，高速化を達成できる．さらに，並列化によって，利用できるメモリが，逐次探索よりもはるかに多くなるので，プランナーの解答能力が向上する．

2. 逐次プランナーの概要

本論文では，現状で最も有効な方法である [Helmert 07] に基づく探索を用いたプランナーを並列化する．この手法では，最初に SAS$^+$ 表現 [BäckStröm 95] で定義できる探索空間を抽象化し，ヒューリスティック関数 $h(n)$ を作成する．次に，$h(n)$ を利用した A* アルゴリズム [Hart 68] によって，プランを求める．[Helmert 07] によって自動生成された $h(n)$ は，A* が最適解の保証のために必要な楽観性と単調性を保持している．

A* は，オープンリストとクローズドリストを利用して節点の展開を行うアルゴリズムである．オープンリストには，未展開の節点を保持する．一方，クローズドリストには，すでに展開した節点を保持する．$g(n)$ を初期状態から，n に至るまでに要したオペレータ数の最小値とし，$f(n) = g(n) + h(n)$ とする．A* では，最初に初期状態をオープンリストに格納する．次に，最適なプランを見つけるか，または目標を達成するプランがないことを証明するまで，$f(S)$ が最小の状態 S を展開し，S をクローズドリストに入れ，S から生成された次の状態をオープンリストに追加する過程を繰り返す．

3. A* 探索の並列化の問題点

本論文では，複数ノードを持つ PC クラスターを利用する．各ノードには，複数コアを持つ CPU があり，これらの CPU コア間ではメモリを共有している．一方，ノード間は，ネットワークで相互に接続されているので，他のノードのメモリにある情報の参照や書き込みの際には，通信が必要である．

A* の並列化の際には，クローズドリストの共有は重要である．プランニングの探索空間の多くは，DAG であるので，別経路から同一状態に至ることがある．クローズドリストによる同一状態の判定を行わなければ，その状態以下を何度も展開してしまい，並列システムの性能低下につながる．

逐次プランナーの A* では，自分自身のメモリ上にあるクローズドリストによって，同一状態の判定が行える．しかし，並列プランナーの A* の場合には，別ノードにある二つのプロセッサが同一状態を生成した場合には，これらのプロセッサが同一の状態を探索している事実の確認は，通信なしには判定できない．さらに，あるノード上のプロセッサ P が他のノード N にある情報 I を参照する際には，通常の実装では，N にあるプロセッサ Q に I の送信を依頼し，Q が I を返すまで，P は I の到着を待つ必要がある．つまり，I の受信のために，P では通信遅延だけでなく，アイドル時間 (同期オーバーヘッド) も生じる．

連絡先: 東京工業大学大学院情報理工学研究科 数理・計算科学専攻〒 152-8552 東京都目黒区大岡山 2-12-1-W8-25
Email:kishimoto@is.titech.ac.jp

表 1: 解答能力

コア数 (メモリ)	1 (16GB)	16 (32GB)	64 (128GB)	128 (256GB)
Pegsol	27	28	29	29
Sokoban	11	16	20	21

4. 並列探索のプランニングへの利用

本研究では，メモリがプロセッサ間で分散している場合でもクローズドリストによる同一節点の判定を行える手法 [Romein 02] に基づいて，プランニングの A* の並列化を行った．この手法のクローズドリストは，各プロセッサは他のプロセッサと互いに素な形でクローズドリストの一部を保持し，全体で一つな大きなクローズドリストとして機能する．本節では，状態 S をクローズドリストに保持するプロセッサを $dest(S)$ とする．オープンリストに関しても同様の実装を行い，各プロセッサは自分自身のオープンリストのみを参照する．

本論文の手法では，各プロセッサは，自分のオープンリストにある最も有望な状態 S を取り出し，S を展開する．S から生成した状態 T は，必ず $dest(T)$ に移動する．さらに，各プロセッサは，他のプロセッサから届いた状態があるかどうかをを定期的に確認し，新たな状態が届いた場合には，自分自身のノードにあるオープンリストに追加する．これらの状態の展開，送受信の過程を，目標を発見するまで繰り返す．本論文の手法の特長は，次のようにまとめられる．

- 別経路で同一状態 S に到達したとしても同じプロセッサ $dest(S)$ に S が割り当てられる．このため，プロセッサ $dest(S)$ が S を探索する必要があるかどうかの確認は，自分自身のメモリ上にあるオープンリストとクローズドリストを調べればよい．
- プランニングでは，[Romein 02] と同様に，各プロセッサへの仕事の分散量をほぼ均等にできる．
- すべての節点の送受信は，非同期に行えるので，ネットワークが遅い環境でも通信遅延の影響を受けにくい．
- より多くの計算ノードがあれば，クローズドリスト・オープンリストをより大きくできるので，メモリのスケーラビリティが良い．

5. 実験結果

前節で説明した A* の並列化法を実装し，国際会議 ICAPS で毎年恒例のプランニング大会 (IPC) の問題集を実際に解かせ，並列探索の性能を評価した．実験には様々な問題を利用したが，ほとんどの問題集で似たような傾向が得られたため，本論文では IPC-6 の Pegsol と Sokoban の各 30 題を利用した．性能の評価は，東京工業大学の TSUBAME 上で行った．TSUBAME の各ノードには Sun Fire X4600 があり，ノードあたり 32GB のメモリと 16 コアの CPU がある．今回の実験では，128CPU コアまで利用した．

表 1，各コア数における解けた問題数を示す．括弧内の数値は利用メモリの合計量を表す．逐次・並列探索で解けない問題は，A* がメモリを使い切ったことを表す．並列化による利用可能なメモリの増大による解答能力の向上が確認できる．

表 2 は，逐次探索にとって難しい問題に対する高速化率である．逐次探索との高速化率の比較をより多くの難問で行えるように，128GB のメモリを持つ特殊な計算ノードで逐次探索の実行を行った．この環境で，逐次探索の求解に 5 分以上 30 分以内の時間制限で解けた問題 (Pegsol 1 題，Sokoban 5 題) を性能比較の対象にした．この特殊ノードは，他ノードよりもクロック数の大きな CPU である (並列探索用のノードは 2.4GHz，特殊ノードは 2.6GHz) ことから，実際の実行時間に $\frac{13}{12}$ 倍を乗じた値を記した．ヒューリスティック抽出に関する並列化は未実装であるが，これに要した時間も含めている．実行時間の隣の括弧内の数値が高速化率である．この表より，本研究の手法は 64CPU コアまでの高速化率は高いことが分かる．

表 2: 合計実行時間 (秒)(1 コアでは実行時間 $\times\frac{13}{12}$)

CPU コア数	1	16	64	128
Pegsol	887	77 (11.5)	24 (37.0)	18 (49.3)
Sokoban	3,928	350 (11.2)	130 (30.2)	103 (38.1)

表 3: 64CPU コアでの難問における合計実行時間 (秒) の比較

CPU コア数	64	128
Pegsol	124	92 (1.35)
Sokoban	393	234 (1.7)

Sokoban での 128CPU コアの高速化率の頭打ちは，多数の CPU コアによる並列探索では，比較した問題自体が簡単に解けてしまうため，すでに並列化による効果を得にくいことが考えられる．図 3 に 64 コアで初めて解けた問題 (Pegsol 1 題，Sokoban4 題) に関する合計実行時間を示す．Sokoban では，引き続き高い速度改善を達成している．

6. まとめと今後の課題

本論文では，[Helmert 07] に基づくプランナーを PC クラスター上で並列化した．その結果，PC クラスターのメモリを全く共有することなしに，高い高速化率が得られただけでなく，プランナーの解答能力も向上した．今後の課題は，ノード内の CPU コアがメモリを共有できることによる更なる高速化や，ヒューリスティック抽出の並列化である．

参考文献

[BäckStröm 95] BäckStröm, C. and Nebel, B.: Complexity Results for SAS^+ Planning, *Computational Intelligence*, Vol. 11, pp. 625–656 (1995)

[Hart 68] Hart, P. E., Nilsson, N. J., and Raphael, B.: A formal basis for the heuristic determination of minimum cost paths, *IEEE Transactions on Systems Science and Cybernetics*, Vol. 4, No. 2, pp. 100–107 (1968)

[Helmert 07] Helmert, M., Haslum, P., and Hoffmann, J.: Flexible Abstraction Heuristics for Optimal Sequential Planning, in *17th International Conference on Automated Planning and Scheduling*, pp. 176–183 (2007)

[Romein 02] Romein, J. W., Bal, H. E., Schaeffer, J., and Plaat, A.: A Performance Analysis of Transposition-Table-Driven Work Scheduling in Distributed Search, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 13, No. 5, pp. 447–459 (2002)