非同期自律分散モバイルロボット群における排他制御とその効率化

森 将真 †　　田村 康将 †　　Défago Xavier†

近年，倉庫や工場などにおいて，モバイルロボット群を用いたシステムの需要が高まってきている．複数ロボットを用いたシステムでは，ロボット同士の衝突を避けるメカニズムが重要となる．スケーラビリティやロバスト性の観点から，こうした衝突回避などを自律分散的に制御する方法論について，多くの研究が行われている．自律分散モバイルロボット群を用いた実環境においては，各ロボットが同期的に移動することは難しい上，通信の遅延も発生する．したがって，ロボットの移動・通信の非同期性を仮定することは実際の応用を考える上で特に重要となる．本研究では，非同期自律分散モバイルロボット群において，各ロボットに任意の経路が与えられたときに，全てのロボットが衝突を起こさずに与えられた経路上を移動するための排他制御をロボット間通信を用いて実現する方法を提案する．また，排他制御の効率化の手法として，限られた範囲の位置のみをロックするウィンドウ型ロックを提案する．シミュレーションによる評価実験の結果，ロックする範囲をロボット数によって変化させることで，ロボットの停止時間が減り，システムにおけるタスクの完了数が増えることが明らかとなった．

Key words: モバイルロボット群，非同期分散システム，排他制御，衝突回避，搬送システム

1. 序　　論

自律分散モバイルロボット群を用いたシステムでは，中央制御機構を持たない複数台のロボットが互いに協調することで，与えられたタスクを実行する．中央制御機構を持ったシステムと比較して，単一故障点が存在しない，スケーラビリティやロバスト性に優れている等，複数の利点が存在する [1]．

モバイルロボット群を用いたシステムの一つとして搬送システムが挙げられる．搬送システムとは，例えば工場のようなところで，物をある地点からある地点に繰り返し運ぶことを目的としたシステムである．実際に Amazon の倉庫ではロボット群によって荷物の管理が行われている [2]．倉庫の他にも，オフィスロボット [3] や空港における航空機の牽引車 [4] などのシステムについて研究が行われている．こうしたシステムにおいて，各ロボットは衝突を起こすことなくタスクを実行することが一般に求められる．

中央集権型のシステムにおいては，ロボット同士の衝突を避ける効率的な経路を計算する手法がこれまでに数多く提案されてきた [5][6][7]．一方で，自律分散的な手法においては，いかにしてロボット群を協調させれば，効率の良い方法でタスクを実行できるかが課題となっている．

ロボット間の協調を考える際に，システムの同期性は重要な要素となる (**Table 1** 参照)．これまでの研究では，同期的なシステムをモデルとした経路計画のアルゴリズムが複数提案されてきた [8][9][10]．しかし，実世界での応用を考えるとき，通信の遅延やロボットの個体差などにより，ロボット同士の通信・移動の完全な同期を仮定することは難しい．また，同期環境を仮定した方法論が非同期環境にそのまま適用可能であることは非常に稀有である．したがって，非同期的にロボットが通信し動くことを前提としてアルゴリズムを設計することが，実ロボットに実装する上で重要である．

Table**1** 同期・非同期システムの特徴.

	同期	非同期
通信	遅延なし	有限時間で遅延あり
移動	同時に動き出し 同時に停止可能	任意時刻において各ロボットが 動いているか止まっているかは任意

† 東京工業大学 情報理工学院

通信・移動の非同期性を仮定したシステムにおける問題として，各ロボットが任意時刻にどの位置にいるかを把握することが難しいため，それぞれのロボットに対し衝突がない効率の良い経路を事前に計算することが現実的でないことが挙げられる．そこで，衝突を避ける経路を事前に計算するのではなく，与えられた経路を各ロボットが衝突なしに進めるよう，システムを協調させる必要がある．

本研究では，全てのロボットが共有する情報を持たないような非同期のシステムを想定する．したがって，メッセージ交換などによる合意形成の手段が必要となる．そこで，ロボットが移動可能な位置を資源と見なし，メッセージ交換による排他制御を通して資源の同時取得，すなわちロボットの衝突が起こらない方法を考える．具体的には，各ロボットが移動先へのロックをリクエストし，ロックに成功した移動先にのみ移動することで，これを実現する．このとき，通信の遅延により，メッセージの送信順序と受信順序が一致しない状況を考慮する．また，各ロボットが一度に取得できるロック数に上限を設けるウィンドウ型ロックを提案し，長時間使用しない移動先をロックし続ける事態を避け，タスク処理の効率化を図る．

搬送システムをモデル化したシミュレーション実験から，ウィンドウ型ロックにおいて，ロボット数に応じてロック数を変化させることで，システム全体におけるロボットの平均停止時間が減少し，タスク完了数を増やせることが明らかになった．

2. 問題設定

2.1 環境モデル

ロボットを動かす環境として，**Fig. 1** のようなグリッド空間を考える．各交点がノード，ノード間の線分がエッジとなる．ロボットはノードからノードへ，エッジ上を通ることで移動するものとする．ロボットはノード上に停止することはあるが，エッジ上に停止することはないものとする．

全てのノードは 2 次元の座標 (x, y) を持つ．このとき，ノード $v_a(x_0, y_0)$ からノード $v_b(x_L, y_L)$ へのパスを以下のように定義する．

$$\pi_{v_a v_b} = ((x_0, y_0), \cdots, (x_n, y_n), \cdots, (x_L, y_L)) \quad (1)$$

ここで，L はパスの経路長を表す[*1]．n はノードの順序番号を表し，ノード (x_n, y_n) とノード (x_{n+1}, y_{n+1}) は互いに隣り合っていることを表す．

本研究では簡単のため，エッジの長さは全て同じと仮定する．

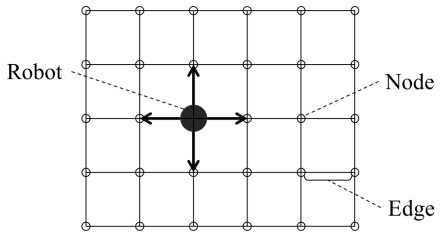


Fig.1 環境モデル．グリッド状の長方形のグラフで，各交点をノード，ノード間をエッジと呼ぶ．ロボットはエッジを通ることでノード間を移動し，ノード上でのみ停止することができる．以降の図では簡単のため，ノードの円は省略する．

2.2 ロボットモデル

各ロボットは他のロボットと通信することで互いの情報を交換する．任意のロボットは他の全てのロボットに直接通信を取ることができると仮定する．ロボットははじめにロボットの総数 N と，一意で不変の ID を与えられるとし，メッセージに ID を付与することで互いを識別する．ロボット同士の通信は非同期であることを仮定する．すなわち，各ロボットは任意のタイミングでメッセージを送信することができ，通信の遅延は有限時間内で存在する．そのため，メッセージは送信した順に受信される保証はない．ただし，メッセージが損失することはないものとする．

ロボットの移動も非同期性を持つ．つまり，あるロボットが動いているとき，他のロボットの位置 (ノード上 or エッジ上) および行動 (移動中 or 停止中) は任意の状態を取り得る．また，各ロボットは他のロボットと通信を行いながら移動できるとする．

さらに，各ロボットは一定量の記憶領域を持ち，自身の記憶領域への参照および書き込みは任意のタイミングで可能とする．

本研究では，ロボットの大きさは考えず，移動速度は一定とし，方向転換のための回転に要する時間は考慮しない．

2.3 クリティカルセクションの定義

衝突には以下の 2 通りが考えられる．

- 2 台以上のロボットが同時に同じノード上に存在する
- 2 台以上のロボットが同時に同じエッジ上に存在する

これらは，次に進むノードをロックしたときのみエッジ上に進み始めることで同一と見なせる．よって，ノードのロックのみに注目すれば十分であり，ノードをクリティカルセクションとして排他制御を行うことでロボット同士の衝突を避ける．

[*1] v_a 自身は経路長には含めない

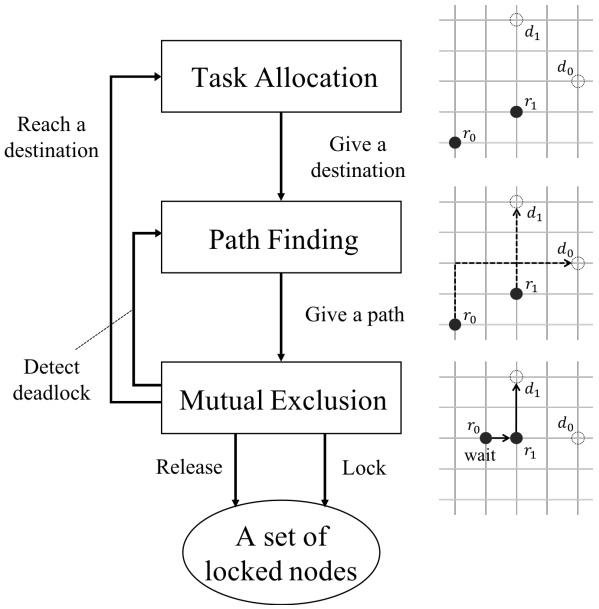


Fig.2 システムの構造．タスク割り当てのアルゴリズムにより各ロボットに目的地が与えられ，経路計画のアルゴリズムによって現在地と目的地を繋ぐパスが出力される．本研究で注目する排他制御のアルゴリズムはそのパスを入力として受け取り，ロックしたノードの集合と，リリースするノードの集合を出力とする．ロボットが目的地に到達したとき，タスク割り当てのアルゴリズムは新たに次の目的地を出力する．d_0, d_1 はそれぞれロボット r_0, r_1 の目的地を表す．

つまり，各ノードは最大で 1 台のロボットによってロックされ，ロックしているロボットのみがそのノードを通ることができる．

また，本研究ではロボットの移動の非同期性を仮定するため，ロボットの数はノードの数よりも必ず少ないという仮定を置く．

2.4 システムの構造

本研究が対象とするシステムでは，タスクは目的地に到達することを意味し，各ロボットにそれぞれ目的地が与えられる．各ロボットは目的地に到達したときに，新たに次の目的地が与えられるとする．

対象とするシステムの構造を **Fig. 2** に示す．

本研究では，排他制御のアルゴリズムに注目し，目的地とそれに至るパスは経路計画のアルゴリズムから入力として与えられると考える．提案手法は，各ロボットに任意の経路が与えられた場合に対して適応可能である．本論文においては，経路計画のアルゴリズムは，グリッド環境において 2 地点を繋ぐ最短距離のパスの集合から，ランダムに一つ選択する．

排他制御のアルゴリズムにおいてデッドロックの解消は重要な課題である．本研究では，経路計画のやり直しによってデッドロックへの対処を行う．詳細は 3.3 節に示す．

3. 非同期分散システムにおける排他制御

3.1 メッセージ交換によるロック

本研究では，ノードを資源と見なし，ロボット間でメッセージ交換による排他制御を行うことで，ロボット同士の衝突を避けることを考える．

V_l^i をロボット r_i がロックしているノードの集合として定義する．r_i は現在居るノード v_c もロックしていると考えるた

め，$v_c \in V_l^i$ であることを注意したい．また，ロボットがエッジ上に居るときは，直前に居たノードを v_c とする．ロボットは V_l^i を監視し，要素があれば順序番号のもっとも小さいノードから移動を開始する．以下では，V_p^{ij} を r_i のリクエストに対し，ロボット r_j がパーミッションを与えたノードの集合として定義する．ロボット r_i が目的地までのパスをロックする手順を以下に示す．

1. リクエストするノードを入れるための集合 V_r^i を用意する
2. パスの要素の中でロックしたいノードを V_r^i に加える
3. リクエストを V_r^i とともにブロードキャストする
4. 他の全てのロボットからのリプライを待機する
5. $\{v \mid \forall j\ s.t.\ j \neq i,\ v \in V_p^{ij}\}$ を V_l^i に加える
6. V_l^i に加えた要素を V_r^i から削除する
7. 目的地のノードが V_l^i に含まれていなければ 2 に戻る
8. 目的地のノードが V_l^i に含まれていれば終了する

メッセージ交換を行う際の通信遅延に対処するため，Last Request Date (lrd) を導入する．また，lrd でリクエストを差別化できない場合には，優先順位を用いることで，処理するリクエストを決定する．詳細は 3.2 節で示す．

以下に，ロボット r_j がロボット r_i からリクエストを受け取ったときの手順を示す．

1. r_j がパーミッションを与えるノードを入れるための集合 V_p^{ij} を用意する
2. パスの要素がない場合，9 に進む
3. リクエストに含まれるパスの要素の中で，順序番号がもっとも小さいノード v_m を取り出し，パスから v_m を削除する
4. $v_m \notin V_l^j \wedge v_m \notin V_r^j$ の場合，V_p^{ij} に v_m を追加し 2 に戻る
5. $v_m \in V_l^j$ の場合，9 に進む
6. $v_m \in V_r^j$ の場合，lrd_j とリクエストに含まれる lrd_i の大小を比較し，$lrd_j > lrd_i$ のときは V_p^{ij} に v_m を追加し 2 に戻り，$lrd_j < lrd_i$ のときは 9 に進む (3.2 節参照)
7. r_j と r_i の優先順位を計算する
8. r_j の優先順位が低い場合，V_p^{ij} に v_m を追加し 2 に戻る
9. V_p^{ij} をリプライに載せ r_i に送り終了する

なお，以降では特定のロボットに明示的に言及することなく，V_l, V_r, V_p の集合を用いる．

3.2 Last Request Date

ロボット同士の通信の遅延は有限時間で存在するため，メッセージの送信順序と受信順序が一致しない状況があり得る．そこで，リクエストを正しく処理するために Last Request Date (lrd) という概念を導入する．

本研究における lrd の考えは，排他制御のアルゴリズムの一つである，Ricart-Agrawala Algorithm [11] に由来する．Ricart-Agrawala Algorithm では単一のクリティカルセクションのみを扱っていたのに対し，本研究では複数のクリティカルセクション (ノードの集合) に対応するため，自分の方が優先順位が低い場合でも，すぐにリプライを送るようにしている．

Fig. 3 (a) のような例では，lrd の導入なしには通信の遅延が原因となり，2 台のロボットが同時に一つのノードのロックをしてしまうため，排他制御が失敗する．

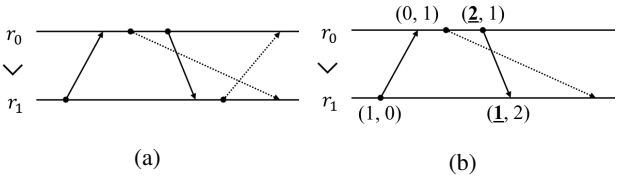


Fig.3 ロボット r_0 とロボット r_1 がノードのロックのために通信する様子．r_0 と r_1 のラベルを持つ横線はロボットの時間軸を表し，左側から右側に時間が経過する．時間軸を行き来する実線はある単一ノードに対するリクエスト，点線はそれに対するパーミッションをそれぞれ表す．r_0 の方が r_1 よりも優先順位が高い．(a) 通信の遅延によって同時に 2 台のロボットが一つのノードのロックをしてしまう例．r_0 が r_1 に送信したパーミッションよりも，それよりあとに送信されたリクエストの方が r_1 に早く届いてしまった場合，優先順位によって r_1 は r_0 に対しパーミッションを送信してしまい，双方がパーミッションを得てしまう．(b) lrd と $clock$ を導入した場合．時間軸横の組は $(lrd, clock)$ を表す．r_1 は r_0 に対しパーミッションを送らない．

そこで Algorithm 1 のように lrd と，その定義に必要となる $clock$ という値を定義する．

Algorithm 1 Last Request Date (Code for robot r_i)

```
1: lrd_i ← 0
2: clock_i ← 0

3: when broadcast REQUEST
4:     lrd_i ← clock_i + 1
5:     Broadcast REQUEST(lrd_i)

6: when receive REQUEST(lrd_j) from r_j
7:     clock_i ← max(clock_i, lrd_j)
```

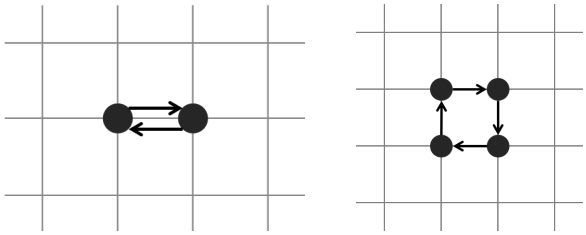
各ロボットはそれぞれの lrd と $clock$ を持ち，ともに 0 で初期化される (Algorithm 1, Line 1, 2)．ロボットがリクエストをブロードキャストするとき，lrd を $clock + 1$ に更新し，リクエストは lrd とともに送る (Algorithm 1, Line 4, 5)．ロボットがリクエストを受け取ったとき，自らの $clock$ の値よりも，送られてきたリクエストに付されている lrd の値の方が大きい場合，$clock$ にその lrd を代入する (Algorithm 1, Line 7)．

ロボット r_j がロボット r_i からノード v に関するリクエストを受け取ったときに，r_j も v をリクエストしていた場合，lrd_j とリクエストに付されている lrd_i の大小を比べる．$lrd_j < lrd_i$ の場合には r_j は r_i にパーミッションを与えず，$lrd_j > lrd_i$ の場合には r_j は r_i にパーミッションを与える．$lrd_j = lrd_i$ の場合には優先順位の比較に移る．優先順位の付け方については様々な方法が考えられ，手法によってもシステムの効率は変わる [9]．本論文においては，実験にて独自の優先順位付けを用いた．

3.3 デッドロックへの対処

モバイルロボット群によるシステムの排他制御において，衝突は第一に避けるべき状況であるが，デッドロックもまた避けるべき状態の一つである．デッドロックの例としては，**Fig. 4** のように 2 台や 4 台のロボットがサイクルを成してしまい硬直状態になる場合が挙げられる．

一般に，デッドロックは以下の 4 つの条件が成り立つときに発生する [12]．



(a) (b)

Fig.4 ロボットがサイクルを形成しデッドロックになる例．(a) 2 台の場合．(b) 4 台の場合．

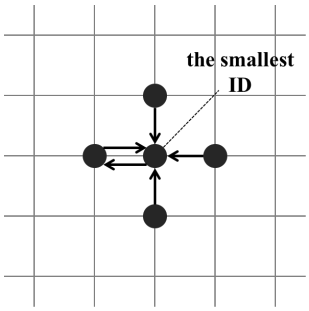


Fig.5 ID の比較のみのときにライブロックが起こる例．中央のロボットがもっとも小さい ID を持つとき，そのロボットが常に退避準備を行うが，退避ノードをどこに選んだとしてもデッドロックが再び生じてしまい，退避することができない．

1. タスクがリソースを占有する
2. タスクが追加で，あるリソースをリクエストしている間，タスクはすでにロックしているリソースを保持する
3. タスクがロックしているリソースはそのタスクの完了に使用されるまで，強制的にロックしているタスクから取り除かれることがない
4. リソースのリクエストがサイクルを形成している

ここで本研究においては，タスクはロボット，リソースはノードに相当する．

これらより，本研究で対象とするモデルにおいては，ロボットが停止しており，停止しているノードの集合に対し，サイクル状にリクエストが行われている場合にデッドロックとなる．

ロボットはリクエストを送る際に，自分の現在地とロックしているノードをリクエストに付す．ロボットは他のロボットからリクエストを受け取ったときに，そのリクエスト内容を記録しておき，新たに同じロボットからリクエストを受け取ったときに記録を更新することで他のロボットがリクエストしているノードの集合やロックしているノードの集合，リクエストが送られた時点での他のロボットの位置を把握する．これらの情報をもとにサイクルが存在するかを調べ，サイクルを形成しているロボットのうち一つがデッドロックを検知する．

デッドロックを解消するには，サイクルを消すように，サイクルを形成しているロボットのうち一つが動けば良い．そこで，ロボット r_i がデッドロックを検知したときにそれを解消する手法として以下のようなアルゴリズムを提案する．

1. r_i がリクエストしているパスの要素の中で，もっとも順序番号が小さいノードをロックしているロボットを r_j とする
2. r_i と r_j のうち，連続で退避準備をしている回数 (連続退避準備回数) が小さい方が退避するノードを選び退避準備をする．連続退避準備回数が同じ場合は，ID が大きい方が退避するノードを選び退避準備をする．退避するノードを退避ノードと呼ぶ．
3. 退避ノードをロックできる場合，退避する．退避ノードのリクエストにより新たにデッドロックが発生した場合，退避準備しているロボットと退避ノードをロックしているロボットとで 2 からやり直す

退避準備をしないもう一方のロボットを待機ロボットと呼ぶことにすると，退避ノードは待機ロボットが存在するノード以外からランダムに選ばれる．連続退避準備回数の比較によって退避するロボットを選ぶことで，**Fig. 5** のような状況のとき ID の比較のみの場合に起こるライブロックを防ぐことができる．

退避したロボットは，退避ノードから目的地までの経路計画をやり直し，新たなパスを受け取る．

4. 排他制御の効率化

4.1 効率化の定義

本章では，モバイルロボット群のシステムにおける排他制御の効率化について説明する．

システムにおける効率化には大きく分けて以下の二つが挙げられる．

- 移動経路長の縮小化
- 停止時間の減少化

本研究では，排他制御の効率化を目的としているため，二つ目の停止時間の減少化について考える．

停止時間とは，ロボットがそれぞれのパスに含まれるノード v に対して，v をロックするために，v 以外のノード上で停止している時間と定義される．すなわち，ロボットがノード上に停止する時間を短くすることが本研究の目的の一つである．

停止時間を少なくするための試みとして，ウィンドウ型ロックを以下に提案する．

4.2 ウィンドウ型ロック

各ロボットは衝突を避けるために，ノードのロックをしてからそのノードに進まなければならない．ここで問題となるのが，一度のリクエストで最大いくつのノードをロックし得るかである．例えば，現在地から目的地までのロックを一度に取得する場合を考える，このとき，現在地から目的地が遠い場合，多くのノードをリクエストするため，他のロボットとリクエストが重複する可能性が高い．また，必要以上に多くのノードをロックするため，他のロボットの動きを妨害し，全体としてロボットの停止時間が長くなると考えられる．

一方で，現在地に隣接するノードのみをロックする場合，隣のノードに移動するたびにリクエストを送り，それに対するリプライを他のロボットから待たなければならない．この場合も通信の遅延などによって停止時間が長くなることが予想される．

そこで，**Fig. 6** のように，最大で決まった経路長のみをロックする方法を導入する．以降，これを「ウィンドウ型ロック」と呼称する．

ウィンドウ型ロックのアルゴリズムを Algorithm 2 に示す．

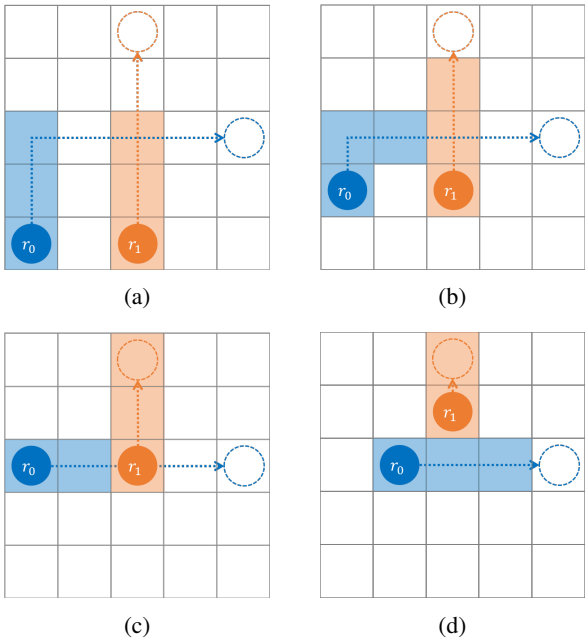


Fig.6 $w = 3$ のときのウィンドウ型ロックを用いた例.

Algorithm 2 では簡単のため，lrd に関する部分は省略している．ウィンドウ型ロックでは，ウィンドウの幅を w としたとき，V_l の要素数と V_r の要素数の和が w 以下になるよう，リクエストを行う (Algorithm 2, Line 5–10)．さらに，ノード v_c のロックは次のノードに移動が完了し v_c が更新されたときに直ちにリリースする (Algorithm 2, Line 33, 34).

5. 評価実験

5.1 評価指標

本研究の目的は，非同期自律分散モバイルロボット群を用いたシステムにおいて，衝突を避けつつ効率良くタスクを遂行することである．ウィンドウ型ロックにおいて，ウィンドウ幅 w がシステムの効率に与える影響を調査するため，搬送システムをモデル化したシミュレーションによる実験を行った．

4.1 節で挙げたように，本研究における排他制御の効率化とは，ロボットの停止時間の減少化を指す．よって，**(a) 一定時間内のロボット 1 台あたりの停止時間**を測ることとする．

また，搬送システムの目的はタスクを完了することであるから，システム全体の効率性の観点から，本実験の評価指標の一つを，**(b) 一定時間内で，ロボット群全体で目的地に到達した回数**と定める．

5.2 実験デザイン

シミュレーションにはプログラミング言語 Java のライブラリ JBotSim[13] を用いた．JBotSim は仮想的に分散アルゴリズムをシミュレーションする環境である．シミュレーションはタイムステップに合わせて進行し，各ロボットは各タイムステップにおいて，移動か停止を選択できる．

本実験において，時間の単位はタイムステップで表されるものとする．実験の指標 (a)(b) において，「一定時間」をこのタイムステップを用いて **10000 タイムステップ**と定義する．また，ロボットの移動速度を 1/1 タイムステップ とし，エッジの長さを 50，すなわち 50 タイムステップで移動可能な距離とした．

Algorithm 2 Windowed Lock (Code for robot r_i)

```
1:  v_dest ← GetDestination(), v_c ← v_init
2:  π ← GetPath(v_init, v_dest)
3:  V_l^i ← ∅, V_r^i ← ∅

4:  while v_c ≠ v_dest do
5:      while |V_l^i| + |V_r^i| < w do
6:          V_r^i ← V_r^i ∪ pop(π)
7:      end while
8:      accepted ← ∞
9:      V_accepted^i ← V_r^i
10:     Broadcast Request(V_r^i)
11:     wait (waiting_from_i = ∅)
12:     V_l^i ← V_l^i ∪ V_accepted^i
13:     V_r^i ← V_r^i \ V_accepted^i
14: end while

15: when receive Request(V_r^j) from r_j
16:     V_p^ji ← ∅
17:     for each v ∈ V_r^j do
18:         prio_i ← Priority()
19:         if prio_i then
20:             break
21:         else
22:             V_p^ji ← V_p^ji ∪ {v}
23:         end if
24:     end for
25:     Send Reply(V_p^ji) to r_j

26: when receive Reply(V_p^ik) from r_k
27:     if |V_p^ik| < accepted then
28:         accepted ← |V_p^ik|
29:         V_accepted^i ← V_p^ik
30:     end if
31:     waiting_from_i ← waiting_from_i \ {k}

32: when reach v_next
33:     V_l^i ← V_l^i \ {v_c}
34:     v_c ← v_next
```

グリッドのサイズは 4×6 とする．グリッドサイズを固定しているが，ロボット数の変化はロボットの密度の変化と見なすことができるため，任意のサイズのグリッドに対しても同様の性質が期待できる．

通信遅延は指数分布にしたがい生成する．本実験では，通信遅延時間の平均 $\bar{d}$ を 10 タイムステップと設定する．

なお，ここで用いられているタイムステップは，分散アルゴリズムを仮想的にシミュレートするために導入されているものであり，この実験においてアルゴリズムの分散性と非同期性は保たれている．

実験目的より，ロボット数 N およびウィンドウ幅 w について，以下の 4×7 パターンで実験を行う．

- ウィンドウ幅 $w \in \{2, 3, 4, 5\}$ (4 パターン)

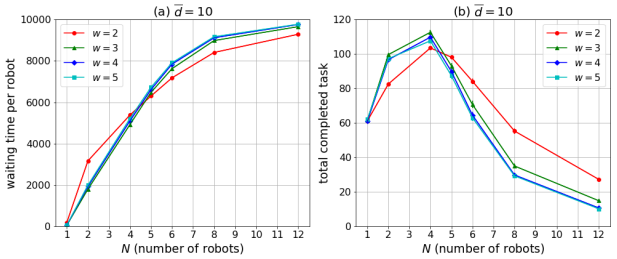


Fig.7 (a) 10000 タイムステップ経過後のロボット 1 台あたりの停止時間．(b) 10000 タイムステップ経過後の完了タスク数の合計．$\overline{d}=10$(固定) で横軸は N．誤差棒は標準誤差を表す．

- ロボット数 $N \in \{1,2,4,5,6,8,12\}$ (7 パターン)

lrd が同じとなったときの優先順位付けは，ロボットからリクエストしているノードまでの距離や連続で同じノードに対しリクエストしている回数，ID をもとに行なった．

シミュレーションは，CPU: 3.1GHz, Intel Core i5, RAM: 16GB の環境において，各条件毎に 100 回ずつ試行した．

5.3 実験結果

Fig. 7 は，10000 タイムステップ経過後のロボット 1 台あたりの停止時間，総完了タスク数のそれぞれについて，100 回試行の平均値を表す．

まずロボット 1 台あたりの停止時間について，N が増加するにしたがって上限の 10000 タイムステップに漸近しながら増加することが分かる (Fig. **7** (a) 参照)．これは N が大きい場合にロボットの停止時間が大きくなることを意味しており，ロボットの密度の増加によりデッドロックが高確率で発生し，その解消に時間を要するためであると考えられる．また，比較的小さい $N(1 \leq N \leq 4)$ に対しては $w=3$，$N \geq 5$ に対しては $w=2$ の場合に停止時間が短くなっており，これは N に応じて適当な w が異なることを示唆する．

Fig. **7** (b) より，完了タスク数については，N がある一定数までは増加し，一定数を超えると急速に減少する．また，N によって完了タスク数を最大化する w が変化していることが分かり，この傾向は停止時間と同様である．

6. 考察

実験結果から，はじめはロボット数が多くなるほど完了タスク数は増加していくが，ロボット数が一定数より多くなると，逆に完了タスク数が減ることが分かった．これはロボット数の増加に伴って，並行で遂行できるタスクが多くなる同時に，混雑により衝突回避やデッドロック解消に要する時間が長くなる性質があり，ある一定数を超えると，後者の効果が前者の効果を上回るためだと考えられる．

N と w の関係については以下のように考察できる．N がある程度小さいうちはリクエストが被る確率が低いため，w を大きめに取ったとしてもロボット間の相互干渉が少なく，全体としての効率は保たれる．一方で，N が大きくなるほどロボットの密度が大きくなるため，他のロボットとリクエストが被る確率が大きくなり，w は比較的小さく与える方が効率が良くなると考えられる．

一方で，どのような N に対しても，$w=4$ と $w=5$ のように，w を一定以上大きくしたところで効率が変わらない現象がある．これは，一度に多くのノードをリクエストしたとしても，他の全てのロボットからパーミッションを与えられたノードのみをロックするという性質から，現在地から一定以上遠くにあるノードがロックできる可能性は少なく，結果的に w を一定以上大きくしても効率が変わらなくなるためと考察できる．

7. 結論

本研究では，非同期自律分散モバイルロボット群における排他制御の実現と，その効率化を目的として，ウィンドウ型ロックを提案した．メッセージ交換によるロックを行うことで，共有資源を持たない非同期分散環境においても，各ロボット同士の衝突を避けることができた．さらに，デッドロックを検知し解消することで，システムが硬直状態に陥ることを防ぐアルゴリズムを考案した．

ウィンドウ型ロックについて，搬送システムをモデル化した実験によってその効果を評価，分析した．実験結果として，ウィンドウの幅をロボット数によって変化させることでシステムの効率が変わることが明らかとなった．

課題として，通信範囲を限定することや，実ロボットを用いた際に発生し得る，通信障害やロボットの故障，ロボット間の速度差などに対応することなどが挙げられる．

謝　辞

本研究は JSPS 科研費 JP17K00019 の助成を受けて行われた．

参 考 文 献

1) Zhi Yan, Nicolas Jouandeau, and Arab Ali Cherif. A survey and analysis of multi-robot coordination. *International Journal of Advanced Robotic Systems*, Vol. 10, No. 12, p. 399, 2013.
2) Peter Wurman, Raffaello D'Andrea, and Mick Mountz. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI Magazine*, Vol. 29, pp. 9–20, 03 2008.
3) Manuela M. Veloso, Joydeep Biswas, Brian Coltin, and Stephanie Rosenthal. Cobots: Robust symbiotic autonomous mobile service robots. In *IJCAI*, 2015.
4) Robert Morris, Corina S. Pasareanu, Kasper SœLuckow, Waqar Malik, Hang Ma, T. K. Satish Kumar, and Sven Koenig. Planning, scheduling and monitoring for airport surface operations. In *AAAI Workshop: Planning for Hybrid Systems*, 2016.
5) David Silver. Cooperative pathfinding. Vol. 1, pp. 117–122. Marina Del Rey, 2005.
6) Trevor Scott Standley. Finding optimal solutions to cooperative pathfinding problems. In *AAAI*, Vol. 1, pp. 28–29. Atlanta, GA, 2010.
7) Guni Sharon, Roni Stern, Ariel Felner, and Nathan R Sturtevant. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence*, Vol. 219, pp. 40–66, 2015.
8) Hang Ma, Jiaoyang Li, T. K. Satish Kumar, and Sven Koenig. Lifelong multi-agent path finding for online pickup and delivery tasks. In *AAMAS*, 2017.
9) Keisuke Okumura, Manao Machida, Xavier Défago, Yasumasa Tamura. Priority inheritance with backtracking for iterative multi-agent path finding, 2019.
10) Prasanna Velagapudi, Katia Sycara, and Paul Scerri. Decentralized prioritized planning in large multirobot teams. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pp. 4603–4609. IEEE, 2010.
11) Glenn Ricart and Ashok K. Agrawala. An optimal algorithm for mutual exclusion in computer networks. *Commun. ACM*, Vol. 24, No. 1, pp. 9–17, January 1981.
12) E. G. Coffman, M. Elphick, and A. Shoshani. System deadlocks. *ACM Comput. Surv.*, Vol. 3, No. 2, pp. 67–78, June 1971.
13) Arnaud Casteigts. The jbotsim library. 2013.