

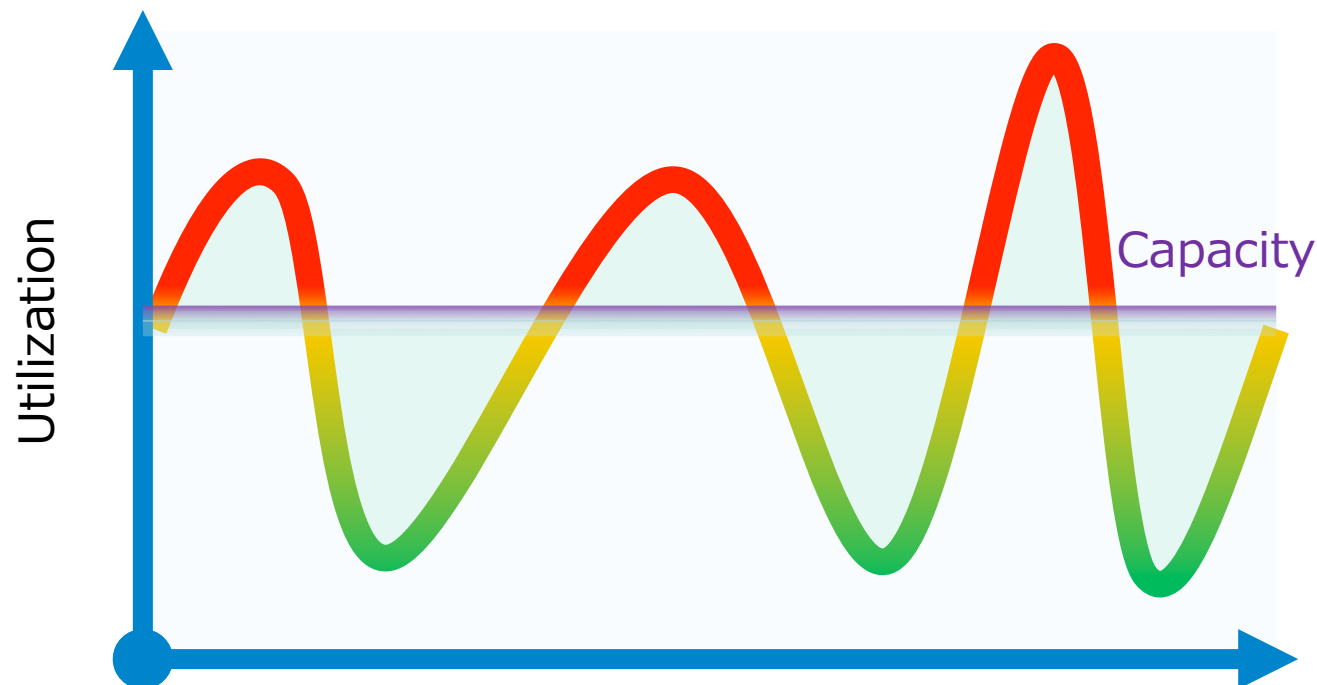
仮想計算機パッキングへの 最適化手法の適用

産業技術総合研究所 情報技術研究部門

中田秀基、竹房あつ子、広淵崇宏、
伊藤智、関口智嗣

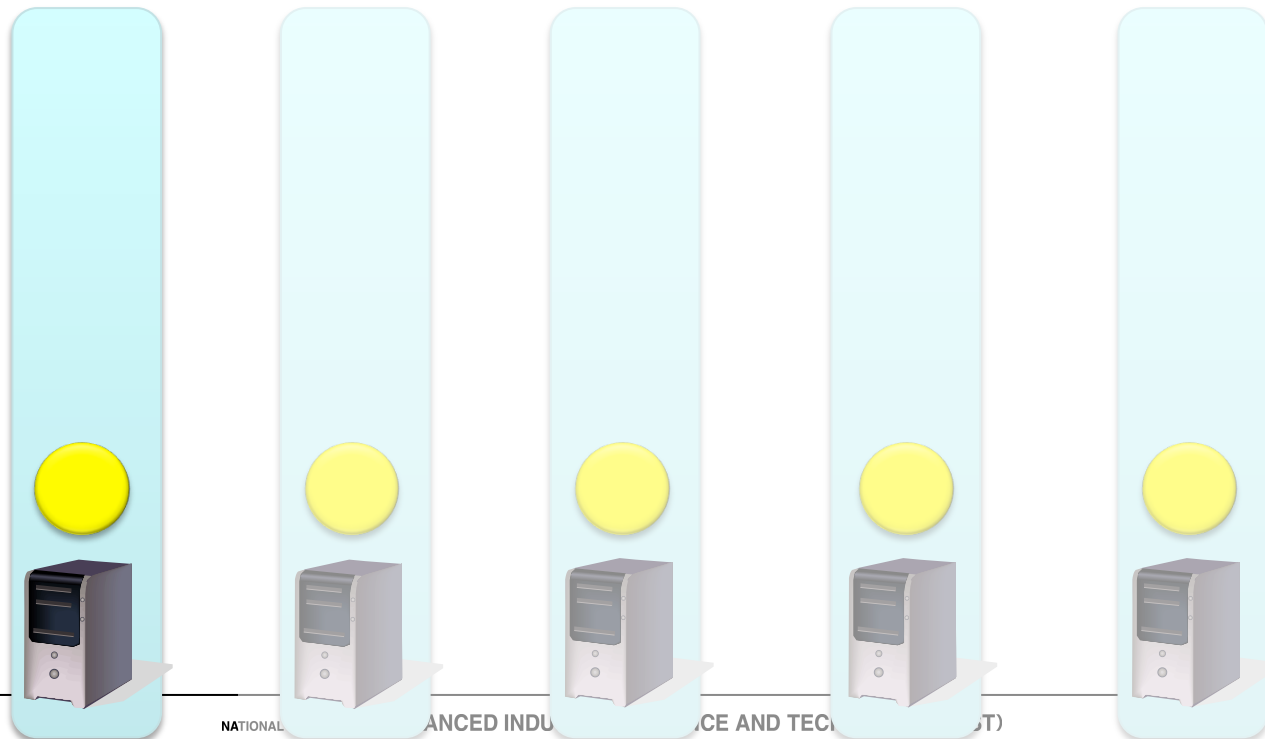
背景

- データセンターの消費電力が増大
- キャパシティプランニング
 - 予想される最大負荷に対してキャパシティを設定せざるを得ない



背景 2

- 仮想計算機による効率的な運用を提案
 - 低負荷時にはライブマイグレーションを用いて下層計算機を片寄せ、ホストをサスペンド
 - 高速マイグレーションを提案 [広渕 OS研究会]



本研究の目的

- 効率的な仮想計算機の配置を決定
 - 仮想計算機パッキング問題
- 仮想計算機パッキングの効率的な解法を探る
 - GAによる解法と整数計画法による解法を提示
 - グリーディな手法と比較

発表の構成

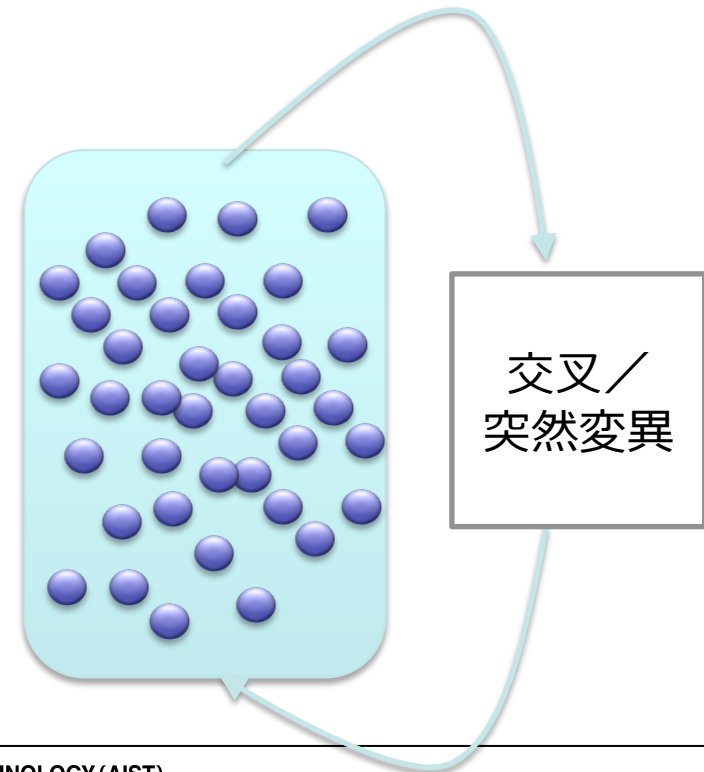
- 仮想計算機パッキングとは
- 遺伝的アルゴリズムによるパッキング
- 整数計画法によるパッキング
- 評価
- まとめと今後の課題

仮想計算機パッキング

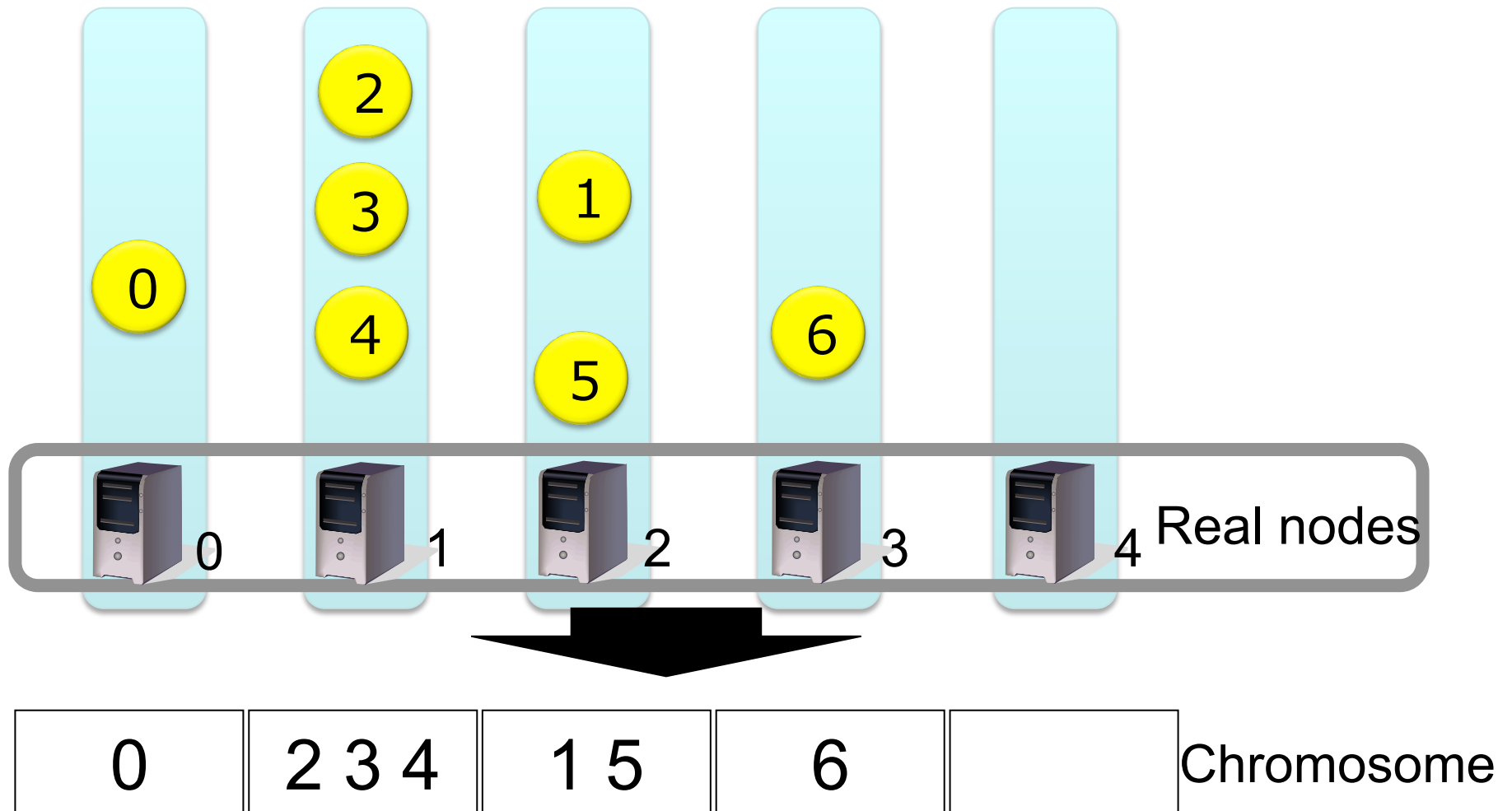
- 目的
 - 使用ホスト数の最小化
 - マイグレーション数の最小化
- 制約
 - あるホスト上で稼働する仮想計算機のリソース使用量の総計が、ホストのキャパシティを超えない
- 困難なポイント
 - ホスト数、仮想計算機数が増大すると組み合わせが爆発する
 - ある瞬間の状態に対して最適化するので、最適化に時間がかかることはできない

遺伝的アルゴリズムによるパッキング

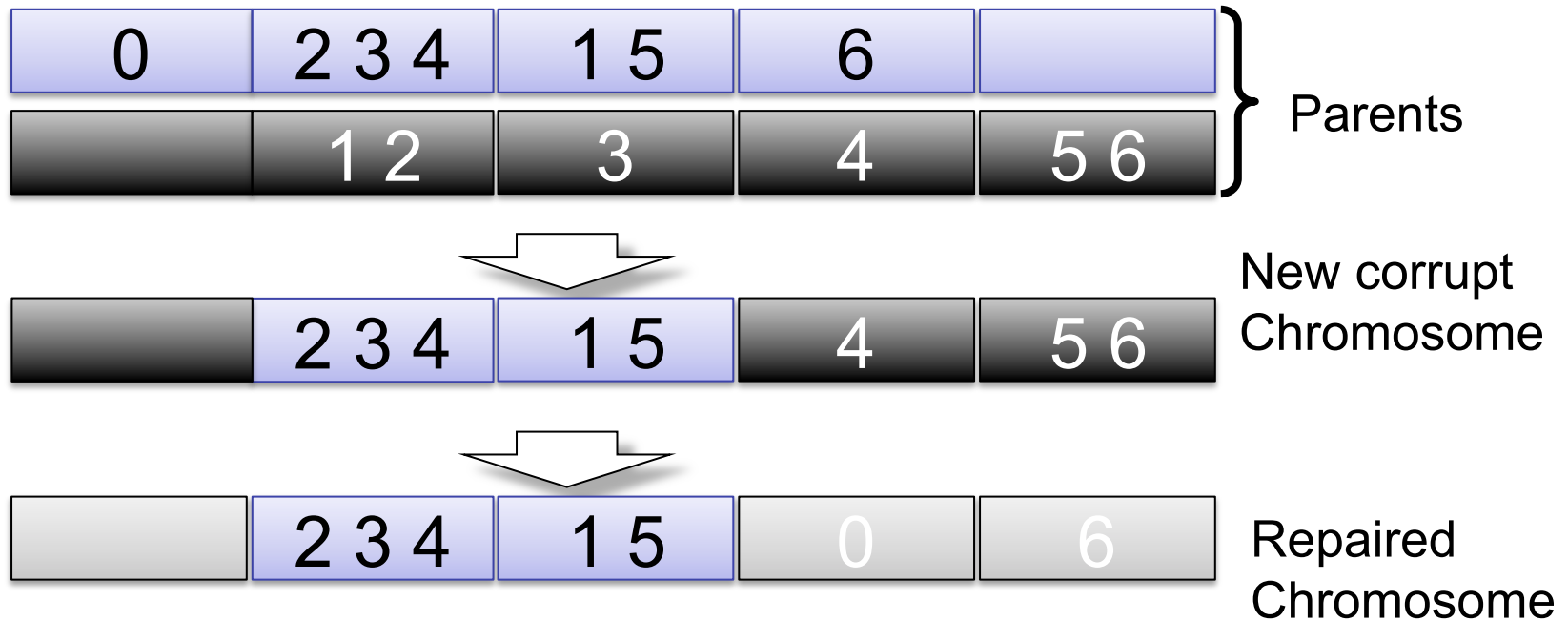
- 遺伝的アルゴリズム
 - メタヒューリスティクスの一つ
 - 多数の解からなるプールを用意し、それらの間の交叉、変異を繰り返すことで、プール内の解を改善
- 遺伝的アルゴリズムの実装
 - 染色体の設計 – 解のエンコード
 - 交叉、突然変異の設計
 - 世代交代の選択



染色体の設計

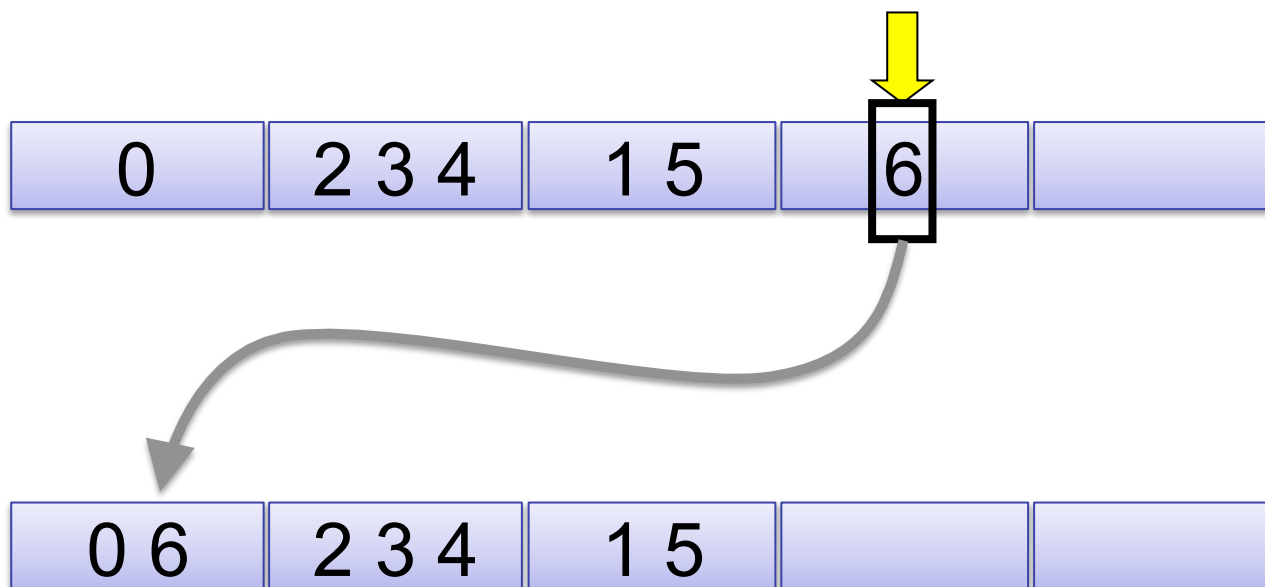


交叉の設計



突然変異

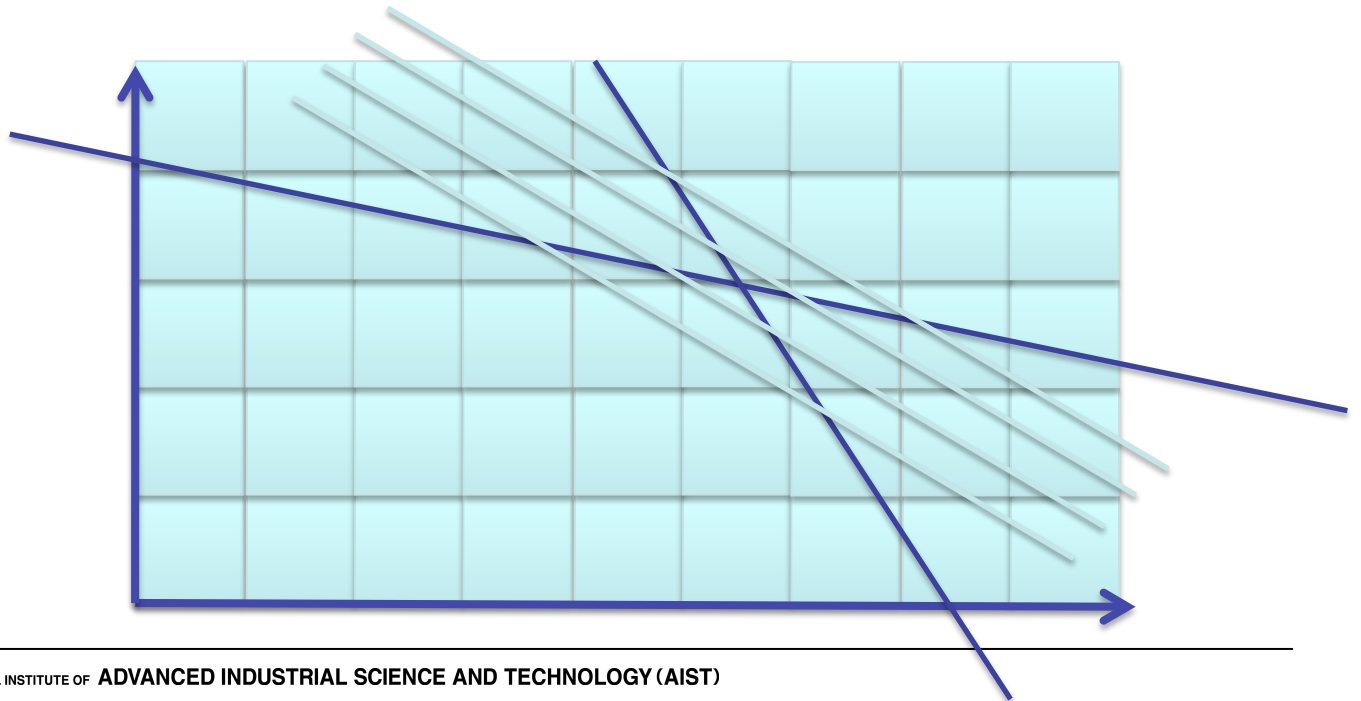
- 任意に仮想計算機を選び、別のホストに挿入



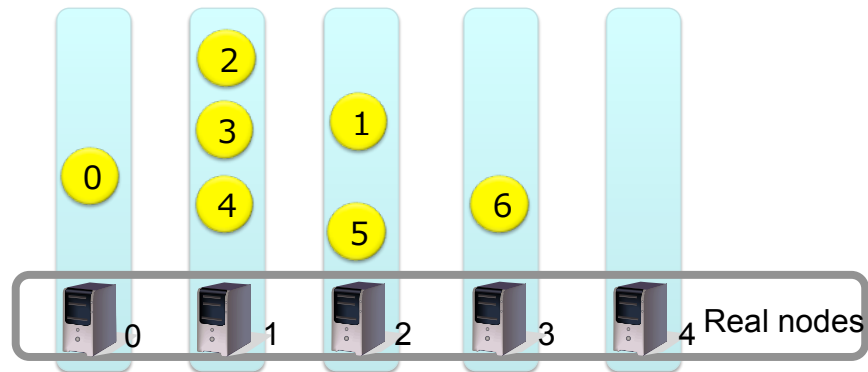
整数計画法によるパッキング

- 整数計画法

- 線形計画法の一種で解の取り得る値を整数に限定したもの
- 一般に線形計画問題よりも困難
- 0-1整数計画法 — 解の取り得る値を0, 1に限定



0-1値で配置を表現

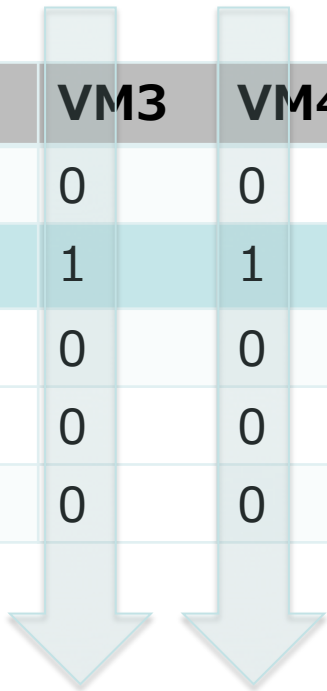


	VM0	VM1	VM2	VM3	VM4	VM5	VM6
Host0	1	0	0	0	0	0	0
Host1	0	0	1	1	1	0	0
Host2	0	1	0	0	0	1	0
Host3	0	0	0	0	0	0	1
Host4	0	0	0	0	0	0	0

制約条件 1

- 1カラムに1は一つ – 1 VMは1ホスト上に存在

	VM0	VM1	VM2	VM3	VM4	VM5	VM6
Host0	1	0	0	0	0	0	0
Host1	0	0	1	1	1	0	0
Host2	0	1	0	0	0	1	0
Host3	0	0	0	0	0	0	1
Host4	0	0	0	0	0	0	0



1
1

制約条件 2

- ホストのキャパシティを超えない

			1M	1M	1M			
		VM0	VM1	VM2	VM3	VM4	VM5	VM6
	Host0	1	0	0	0	0	0	0
4M	Host1	0	0	1	1	1	0	0
	Host2	0	1	0	0	0	1	0
	Host3	0	0	0	0	0	0	1
	Host4	0	0	0	0	0	0	0

4M > 1M+1M+1M

目的関数 1

- 使用ノード数を最小化
- 各ホスト上のVM数をカウント
- ≥ 1 なら 1, $=0$ なら 0 として加算

	VM0	VM1	VM2	VM3	VM4	VM5	VM6	
Host0	1	0	0	0	0	0	0	1
Host1	0	0	1	1	1	0	0	3
Host2	0	1	0	0	0	1	0	1
Host3	0	0	0	0	0	0	1	1
Host4	0	0	0	0	0	0	0	0

1

1

1

1

0

4

目的関数 2

- マイグレーションを最小化
 - =動いていないノード数を最大化

	VM0	VM1	VM2	VM3	VM4	VM5	VM6
Host0	1	0	0	0	0	0	0
Host1	0	0	1	1	1	0	0
Host2	0	1	0	0	0	1	0
Host3	0	0	0	0	0	0	1
Host4	0	0	0	0	0	0	0

	VM0	VM1	VM2	VM3	VM4	VM5	VM6
Host0	1	0	0	0	0	0	0
Host1	0	1	0	0	0	0	0
Host2	0	0	1	0	0	1	0
Host3	0	0	0	1	0	0	1
Host4	0	0	0	0	1	0	0

整数計画法の実装

- GLPK (GNU Linear Programming Kit)
 - フリーの実装
 - 低速であることで知られる
- 問題はGMPL (GNU MathProg Language)で記述
 - 抽象度の高い記述が可能
- GLPKの制御はJavaから
 - GLPK Javaを利用
 - SWIGとJNIを利用したJavaバインディング
 - コマンドラインからでは難しい詳細な制御が可能

GMPLによる記述

```
set VM;
set HOST;
param current{v in VM, h in HOST}, binary;
var location{v in VM, h in HOST}, binary;

param load_requirement{VM}; # 0 - 1.0
param load_capacity{HOST}; # 1.0 ?
param mem_requirement{VM}; # 0 - ? in Mbytes, integer
param mem_capacity{HOST}; # - in Mbytes, integer
var haveVM{HOST}, binary;
var hostCount;
var nonMigrationCount;

minimize nodes: hostCount * 80 - nonMigrationCount;

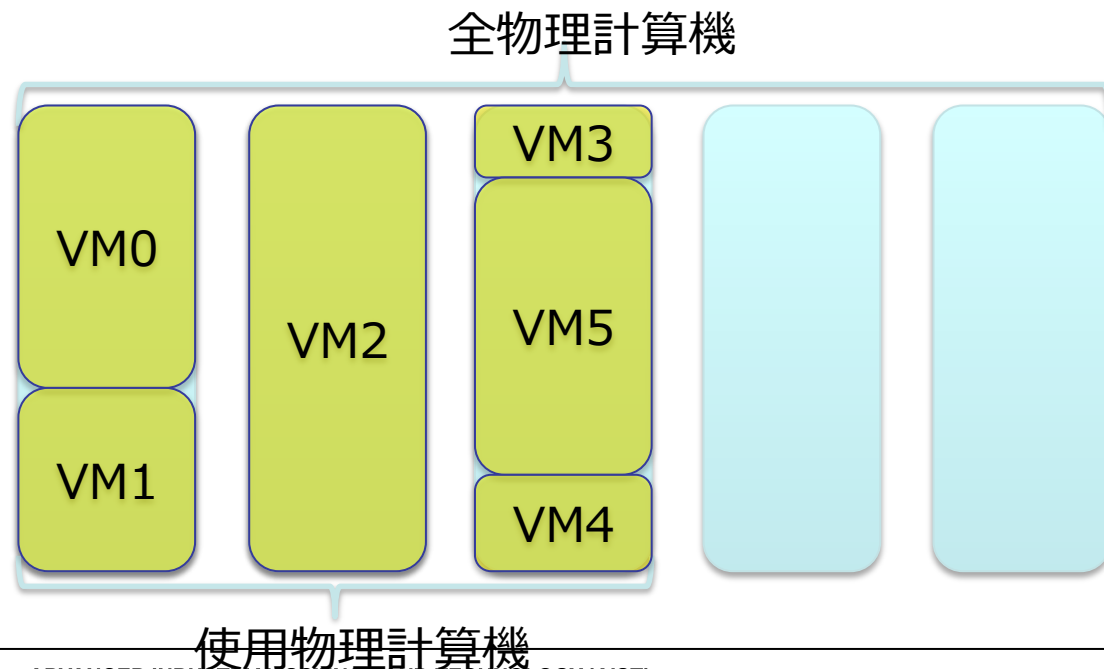
s.t. oneVM{v in VM} : sum{h in HOST} location[v, h] == 1;
s.t. haveVMLimit{h in HOST} : sum{v in VM} location[v, h] <= haveVM[h] * 1000;
s.t. loadLimit{h in HOST} : sum{v in VM} load_requirement[v] * location[v, h] <= load_capacity[h];
s.t. memLimit{h in HOST} : sum{v in VM} mem_requirement[v] * location[v, h] <= mem_capacity[h];
s.t. hostLimit: hostCount = sum{h in HOST} haveVM[h];
s.t. migrationLimit: nonMigrationCount = sum{v in VM, h in HOST} current[v, h] * location[v, h];
```

評価

- 評価問題の生成法
- 評価比較対象：FDD
- GAの評価
- 整数計画法の評価

評価問題の生成

- 使用する物理計算機数を指定、VM数で分割
 - 指定数以下のノード数にパッキングする方法はない
 - 物理計算機数 10, 20, 30, 50, 100
 - 仮想計算機数 – 物理計算機の3倍
 - 使用物理計算機の比率 0.1 .. 0.9
 - 初期配置はサイクリック



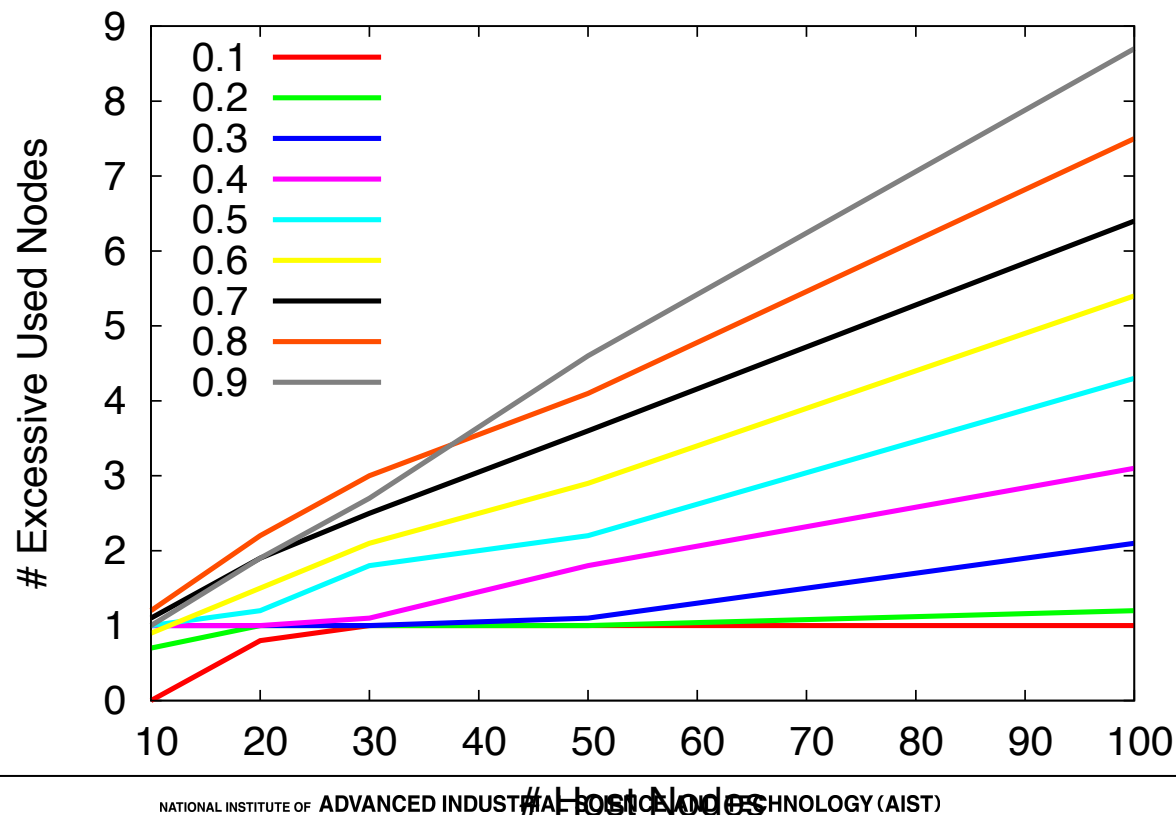
比較基準：FFD

- FFD: First Fit Decreasing
 - Greedyヒューリスティクスの一つ
 - 最適ではないがよい近似解が得られる
 - 高速：最大の問題でも0.1秒以下
- 資源要求の厳しい順番に仮想計算機をソート
 - この順番で先頭のノードから埋めていく
 - 入らなければ次のノードへ



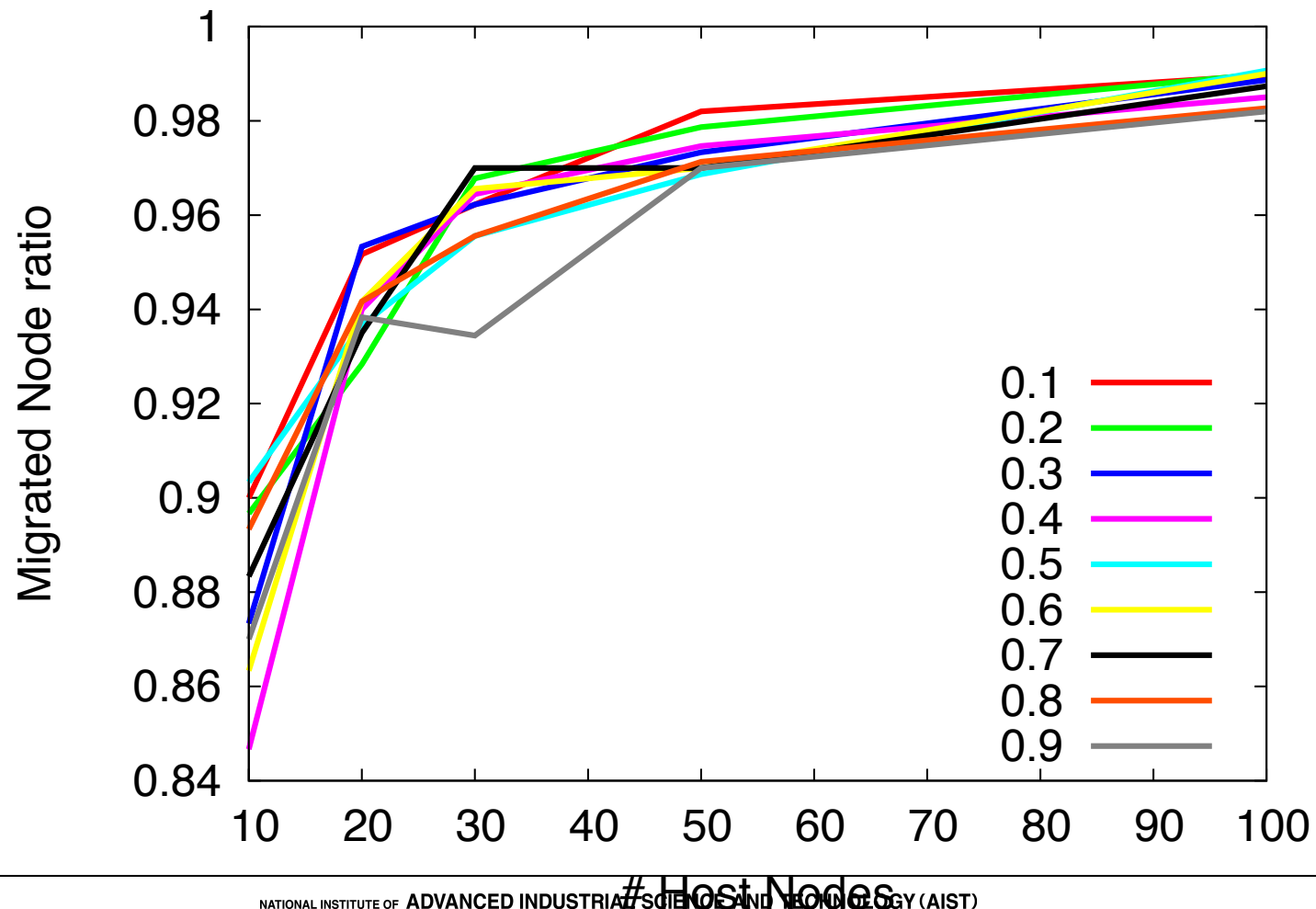
FFDによるパッキング

- 最適解と比較して使用した余分なノード数
 - 例 100台で0.9 – 90台が最適だが、99台使っている



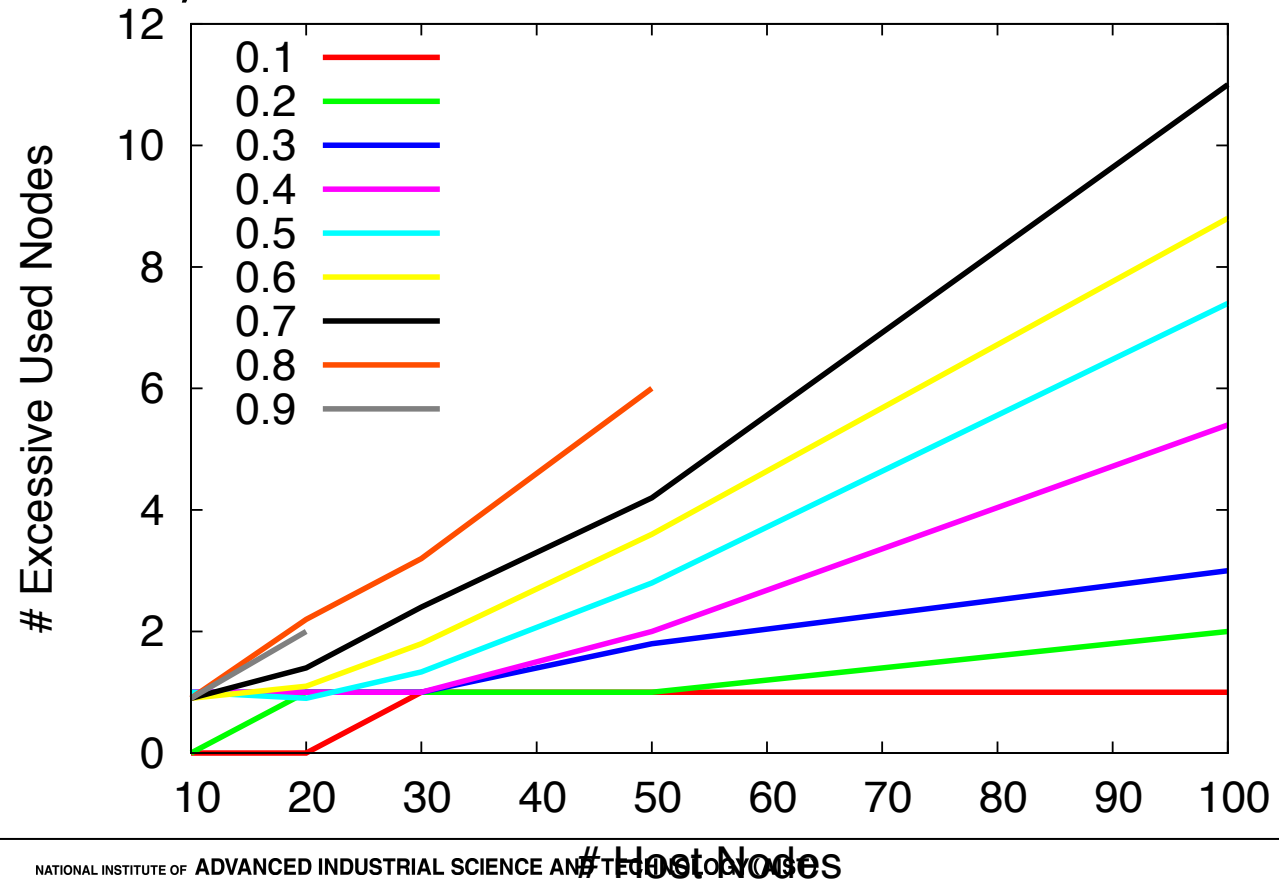
FFDによるマイグレーション

- ほとんどのノードがマイグレートすることに



GAによるパッキング

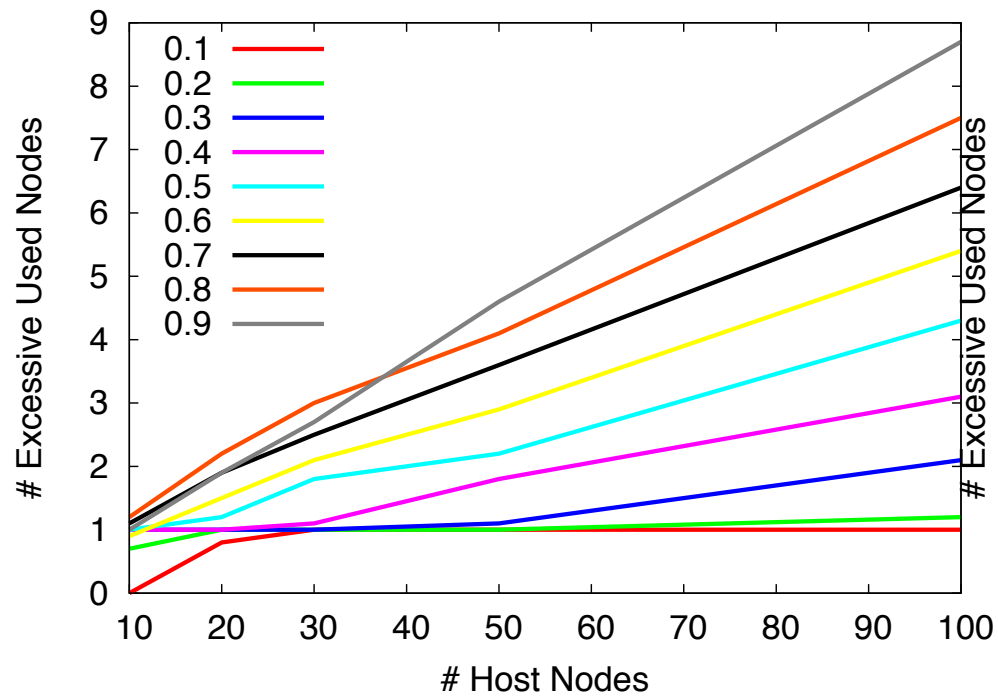
- 300秒間の最適化
- 困難なケースでは可能解が得られない
 - 100台 0.8-0.9, 50台 0.9



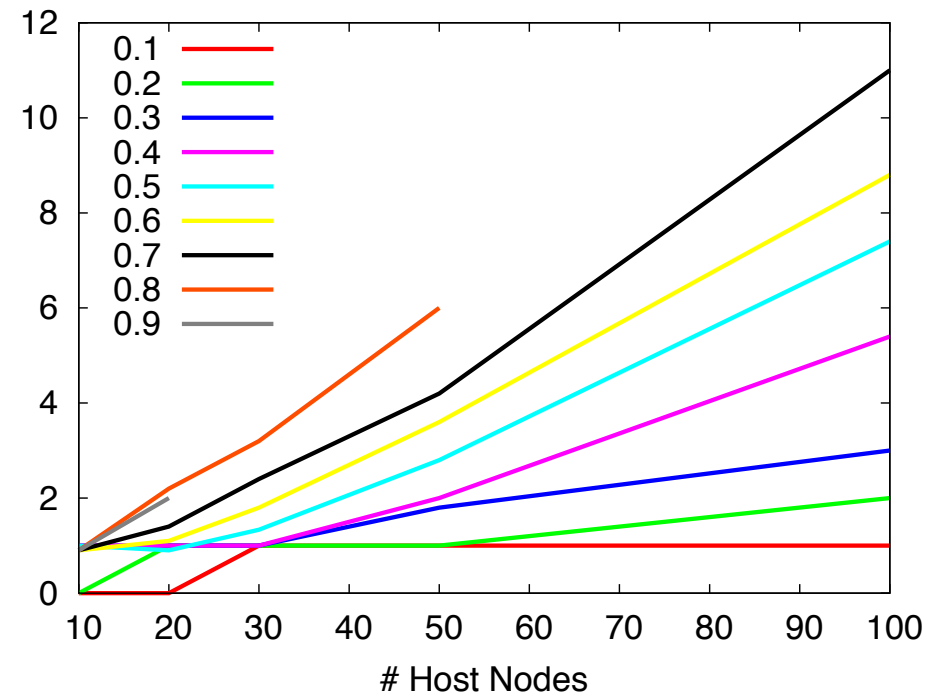
FFD vs. GA

使用ノード数

FFD

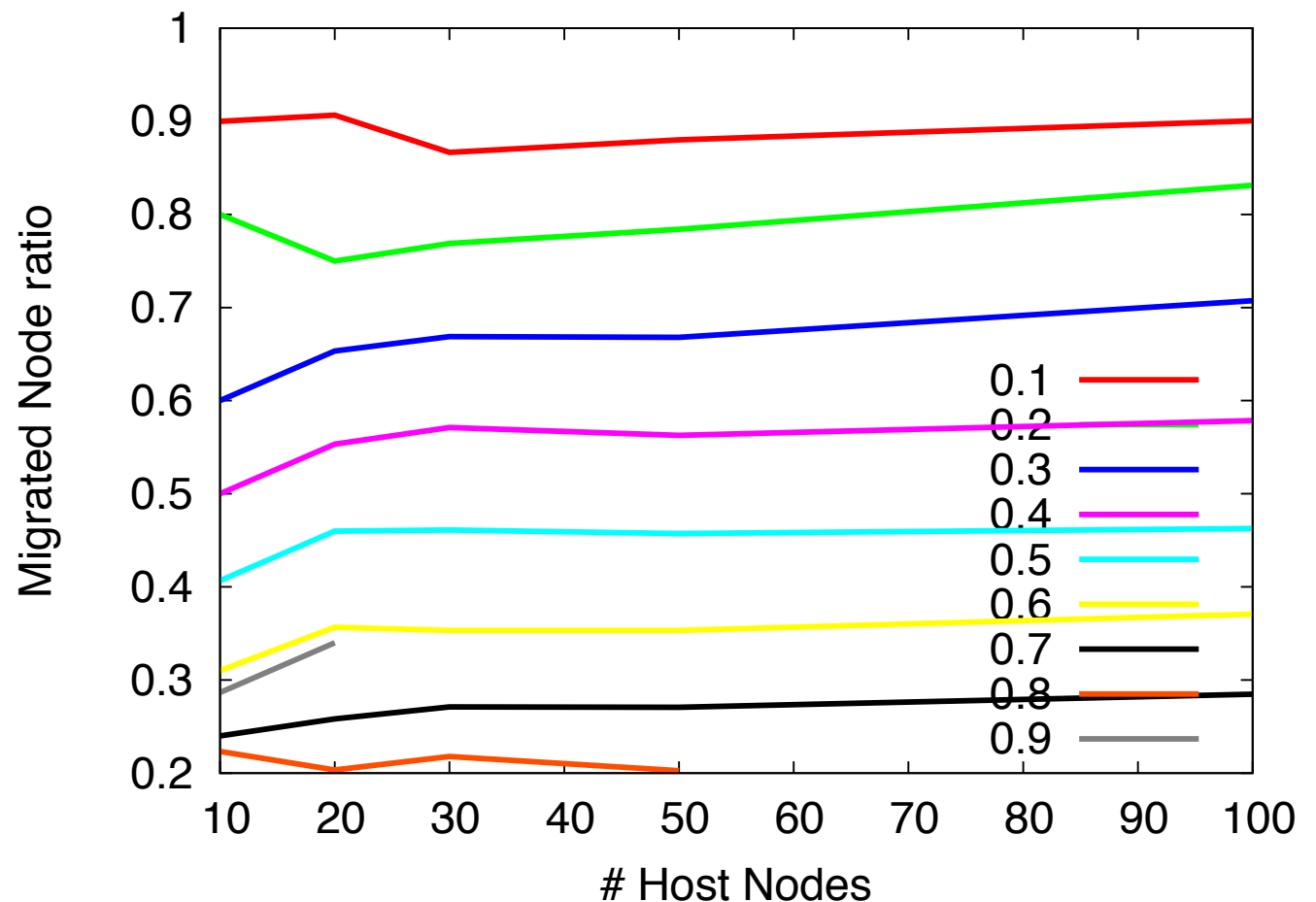


GA



G Aによるマイグレーション

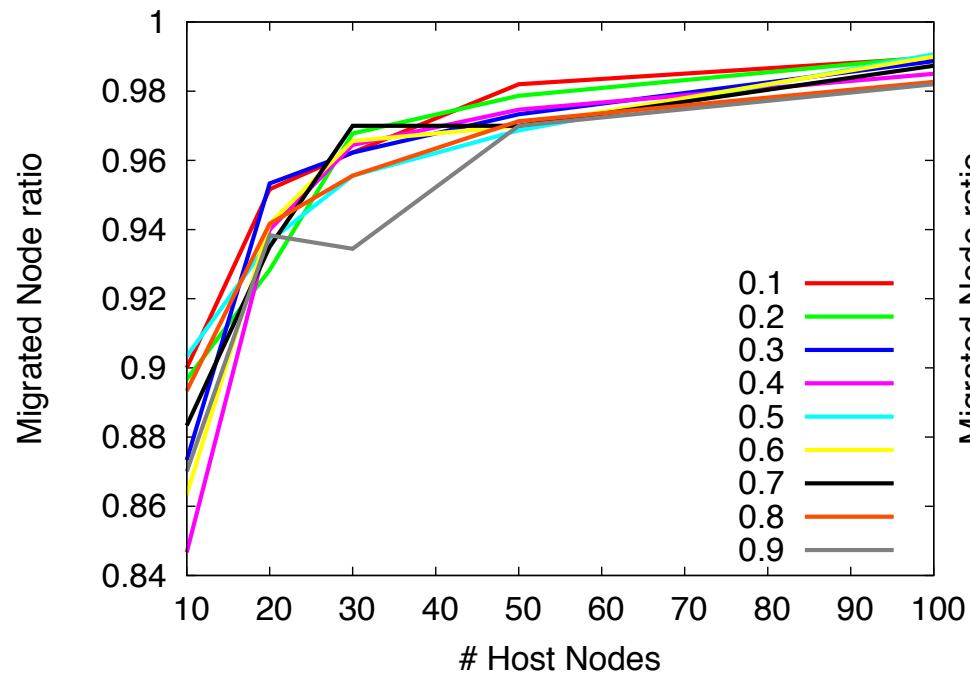
- マイグレーション数が抑制できている



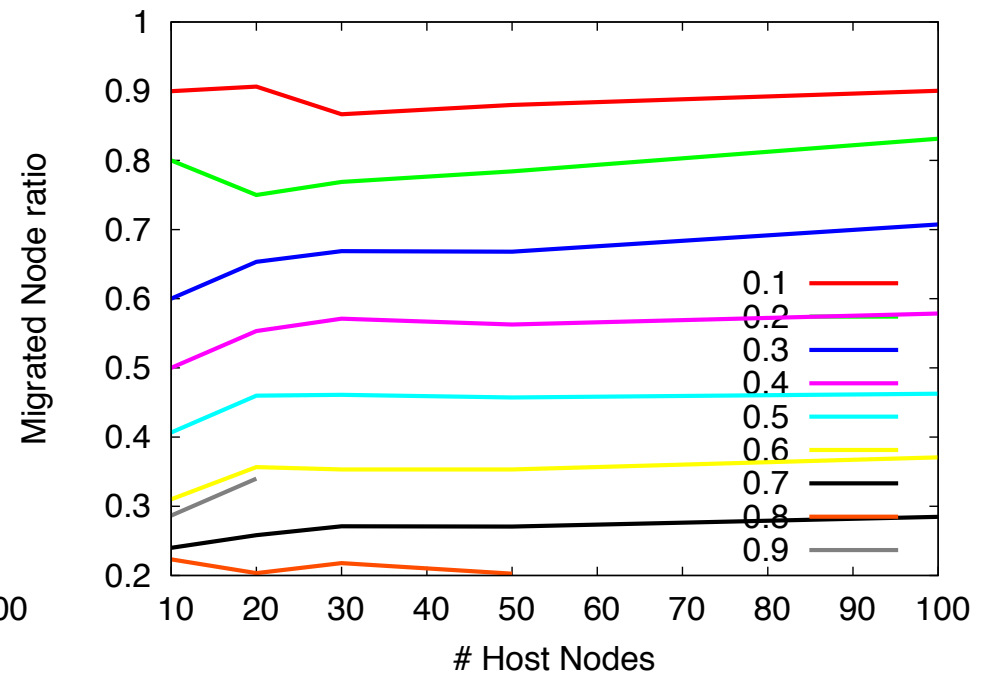
FFD vs. GA

- マイグレーションしたノードの割合

FFD

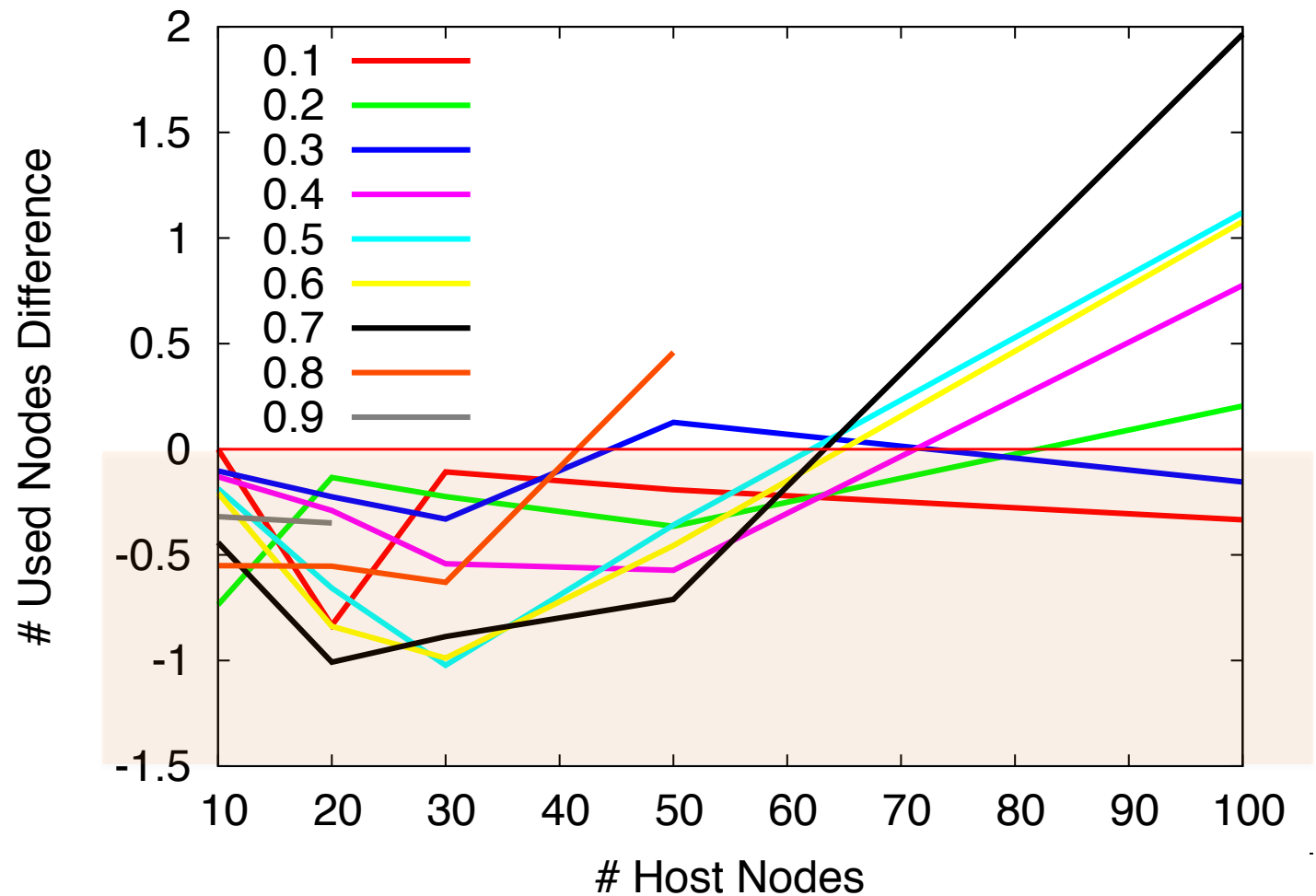


GA



FFD法とGAの比較

- マイグレーション数を重み付けして利用ノード数から引いた結果の比較
- 0以下の部分ではGAがよい結果



GAのまとめ

- 300秒という非現実的な最適化時間を費やしてもFFDと比較してよい解は得られなかった
- マイグレーションに関してはFFDよりも良好な性質が見られた

0-1整数最適化の評価

- 30秒の最適化で可能解が見つかるかどうかを測定
- 乱数のシードを変えて問題を5つ作成し、それぞれに対して実行

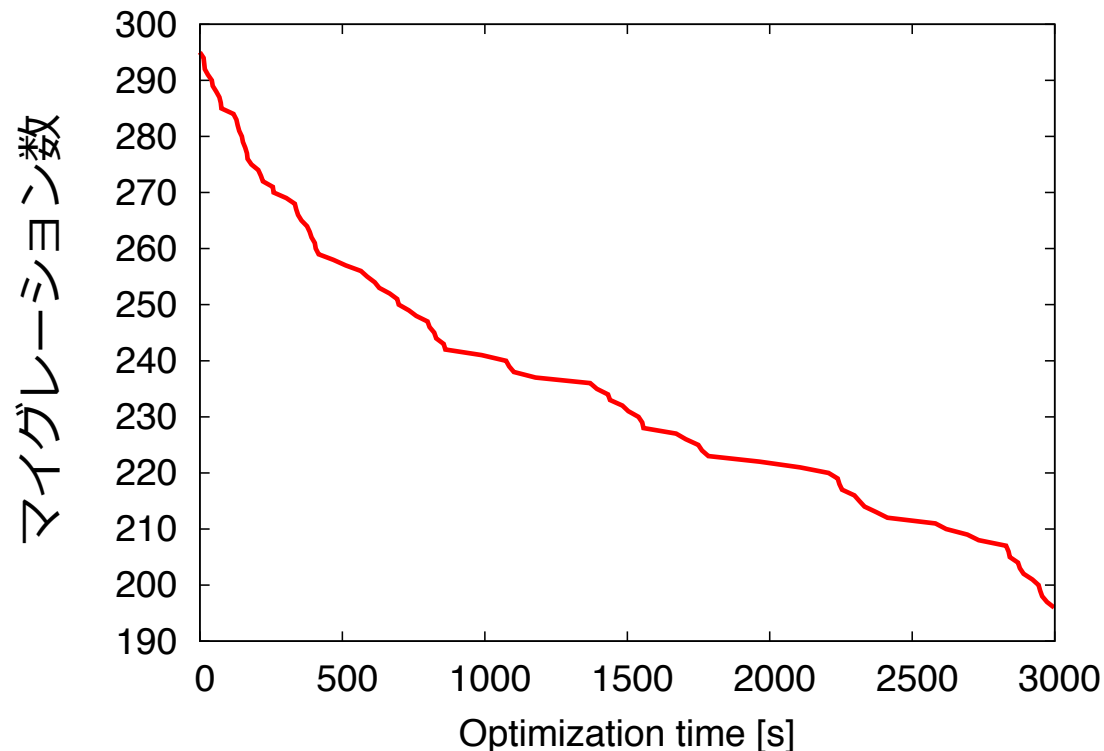
ホスト数\比率	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
10	100	100	100	60	40	0	20	20	0
20	100	0	0	0	0	0	0	0	0
30	0	0	0	0	0	0	0	0	0
50	0	0	0	0	0	0	0	0	0
100	0	0	0	0	0	0	0	0	0

0-1整数最適化法のまとめ

- GLPKを利用し、ナイーブなアプローチで臨む限りでは、実用にならない
- なんらかの工夫が必要

GAによるFFD法解の最適化

- FFD法で得られた暫定解をGAで改善
 - マイグレーション数を改善
 - 時間がかかりすぎ



関連研究

- Entropy [Hermenier 09]
 - 制約解消問題として仮想計算機パッキングを実現
 - 0-1整数計画法と同様のアプローチ
 - 仮想計算機群を、性能に関する要請によってクラス分け
 - クラスに対して配置を決定
 - 最大で8クラスまでしか論文には書かれていない
 - 動的に必要な資源量が変わる場合には対応できない

まとめ

- 仮想計算機パッキング問題
 - GAおよび整数計画法による解法を提示
- FFD法による解法との比較
 - GAも整数計画法も低速、実用は苦しい
 - FFD法はマイグレーションを考慮に入れないためFFDの解の性質も悪い
 - FFDによる解をGAで改良する方法は比較的有望

今後の課題

- 整数計画法を他のソルバで試す
 - GLPKは非常に低速
 - 他のソルバを用いることで高速化を期待
 - CPLEX はGLPKと比較して2桁高速な場合もある
- 制約を加えて計画を高速化
 - 一度のプランニングで数十のVMを同時に動かすプランを生成する必要はない
 - マイグレーション数を限定してプランを生成

謝辞

- 0-1整数計画法でのモデリングに関してご助言いただいた、中央大学藤澤克樹准教授に感謝いたします
- 本研究の一部は、CREST(情報システムの超低消費電力化を目指した技術革新と統合化技術)により実施した