Grid 環境に適した並列組み合わせ最適化システムの提案

秋山智宏[†] 中田秀基[††,†]
松岡聡[†,†††] 関口智嗣[††]

多次元パラメータ関数の最適値を求める組合せ最適化問題の解法としては、分枝限定法や遺伝的アルゴリズムなどが知られている。これらの解法は自明な並列度が大きく、粒度の調整も比較的容易なため Grid 上での実行に適している。しかし Grid 環境での分散並列プログラミングは煩雑である上、実行時にも実行ファイルや設定ファイルをユーザがインストールしなければならないといった問題がある。われわれはこれらの問題を解決し、最適化問題解法の Grid 上での実行を容易にするシステム jPoP を提案する。jPoP は各解法に対してテンプレートとなるクラスを提供しプログラミングを支援する。また、動的なプログラムのアップロードによって Grid 上での実行を支援する。本稿では jPoP の概要と遺伝アルゴリズム問題のテンプレート、さらに jPoP の実装について述べる。

A Proposal for Parallel Combinatorial Optimization System for the Grid

Tomohiro Akiyama ,[†] Hidemoto Nakada ,[††,†]
Satoshi Matsuoka [†,†††] and Satoshi Sekiguchi[††]

For combinatorial optimization problems, which compute the optimal value of a multi-dimensional parameter function, serveral methods are known to be effective, such as Branch-and-Bound methods, Genetic Algorithm, etc. Since these methods can be massively parallelized and the granularities of computation tasks are easily controllable, they are considered to be suitable for executing on the Grid. However, distributed parallel programming on the Grid is quite complicated and furthermore setting up the Grid-wide computing environment is a heavy burden. Here, we propose a system called jPoP, which makes it easy to develop and execute optimization-problem solvers on the Grid. To support the development, the jPoP provides a template class for each algorithm. And to reduce the cost of the setup, it automatically stages the user programs to the Grid environment. This paper describes the design and implementation of the jPoP system. The template classes for Genetic Algorithms are also shown.

1. はじめに

我々は現在 Grid 上のアプリケーションとして、組み合わせ最適化問題[1] に注目している。組み合わせ最適化問題は実社会において、スケジューリング、設計問題、生産計画など広大な応用範囲を持つ問題であり、かつ自明な並列性が大きく実行粒度の調整の自由度も高い。これまでも我々は分枝限定法や遺伝的アルゴリズムに対して Ninf-1 システム[2] を適用し、これらの問題に対する Grid 技術の有効性を確認してきた[3),4)]。

しかし、一般的に Grid アプリケーションを実装することは、1) 広域に分散したアーキテクチャや OS、性能などが異なる計算リソース (PC クラスタ、スパコン等) 群の取り扱い、2)Grid 上の異なるサイト間の安全な通信、リソースの保護が必要、3) 通信、同期、負荷分散などの並列プログラミングの知識が必要、という問題のため困難である。さらに、組み合わせ最適化アプリケーションでは、4) 組み合わせ最適化問題のアルゴリズム、データ構造などを一から分散実装する、という煩雑さがある。

上記に挙げた 1)、2)、3) に関しては、Ninf-1 等の Grid RPC システムを用いることでそれまでの既存の方法に比べて大きく負担が軽減されることが確認されている[3),4)]。しかし、Grid RPC システムを用いてもアルゴリズムプログラマにとっては 4) のような問題は解決されず、その負担は依然として大きい。

この問題を解決するために、本研究では並列組み合わせ最適化システム jPoP を提案する。jPoP は、代表的な数種の並列組み合わせ最適化アルゴリズムのテン

† 東京工業大学 Tokyo Institute of Technology
†† 産業技術総合研究所 National Institute of Advanced Industrial Science and Technology
††† 国立情報学研究所 National Institute of Information

プレートを提供し、プログラムの並列化、安全性などの上記問題をプログラマから隠蔽する。プログラマは問題領域依存なデータやその操作を定義するだけで、Grid 上で組み合わせ最適化アプリケーションを容易に開発でき、かつ安全に実行することができる。

2. 組み合わせ最適化問題

本章では組み合わせ最適化問題の概要について述べる。組み合わせ最適化問題は数理計画問題の一つの分野であり、「与えられた制約条件のもとで、最大の評価が得られるような組み合わせを求める」という問題である。

有名な問題としては巡回セールスマン問題 (TSP) や 2 次割当問題 (QAP)、ナップザック問題などがある[1)]。組み合わせ最適化問題は、スケジューリング、設計問題や生産計画などの実社会において広い応用範囲を持っているが、実社会で求められる大規模な問題を解くには膨大な計算量が必要である。しかし、組み合わせ最適化問題の多くが独立性の高い部分問題に分割しやすいという性質を持つため、並列化によって高速化及び適用問題の大規模化が期待できる。多くのアルゴリズムの中で我々は 3 つの最適化アルゴリズムに注目してる。以下ではそのアルゴリズムと並列化手法について概要を述べる。

遺伝的アルゴリズム (GA) GA は生物の進化プロセスから着想された多点探索に基づく探索アルゴリズムであり、自然淘汰・交差・突然変移等の特徴的な操作を用いて新しい個体 (探索点) を生成し、実用解あるいは最適解を高速に発見する手法である。並列化手法として、一つの母集団を複数の集団に分割し、複数の PE(Processing Element) に割り当てる分散 GA[5)] がある。

分枝限定法 (Branch and Bound) 分枝限定法は問題を複数の部分問題に分割し (分枝操作)、部分問題の中に最適解が見つかるかもしくは最適解が存在しないことを判定し、それ以降の解の探索を取り止める (限定操作) ことで解の探索効率を挙げる手法である。並列分枝限定法[6)] では分割した部分問題を複数 PE に割当て、各々で探索と分枝操作、限定操作を行う。

焼き鈍し法 (SA) 焼き鈍し法は確率的に解候補を改善していく手法である。この手法は、暫定解近傍を調べてより評価値の良い側へ暫定解を変化させる操作を繰り返すことで良質な解を求める。さらに、解の評価値が悪くなる方への移行を確率的に認め、温度という状態遷移確率を制御する変数を高い方から低い方へ徐々に変化させることで最適解を求める。これを複数の異なる温度を別々の PE で並列に行い、隣り合う温度間で定期的に解交換を行うのが温度並列焼き鈍し法[7)](レプリカ交換法) である。

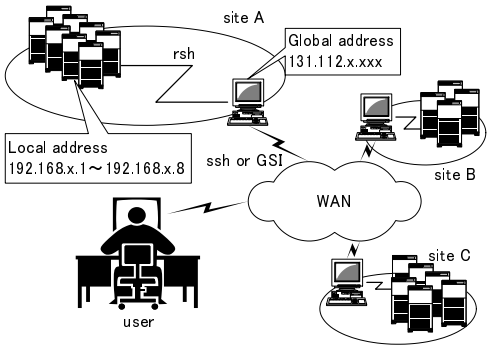

図 1 jPoP の実行環境

3. jPoP の設計

jPoP は Grid 環境として複数の PC クラスタからなる実行環境、クラスタオブクラスタを想定している (**図 1**)。それぞれのクラスタには、グローバルなアドレスを持ち外部と通信が出来るノード (マスターノード) が少なくとも一つあるとし、各クラスタの内部のノードは外部と直接通信することが出来なくても良い。

このような実行環境におけるアプリケーションには以下のような要請がある。

- 任意のプラットフォームでの実行
- 高いセキュリティをもった通信とリソースの保護
- 並列プログラミング

さらに、組み合わせ最適化問題では、

- 並列度のスケーラビリティの獲得が困難

という問題が挙げられる。 既存の Grid 上での組み合わせ最適化アプリケーションはマスター・スレイブ方式の実装が一般であり、これまでの実験ではマスターが一つに対してスレイブが数台から数十台規模のローカルな環境におけるものに過ぎなく、数百台さらには数千台といった大規模な並列環境にスケールできるかどうかは確認できていない☆。

以下では、上記の要請に対して jPoP がとる解決法について述べる。

3.1 プラットフォーム独立性

jPoP は実装に Java を用いることで、さまざまなプラットフォーム上での実行を可能にする。クラスタの各ノード上には Server、マスターノード上には ClusterManager と呼ばれる Java プログラムが存在する。ユーザは、やはり Java で書かれたクライアントプログラムを自分の記述したクラスを引数として起動する。クライアントは ClusterManager を介して Server と

☆ 実際遺伝的アルゴリズムの実験において、単体のマスターに対しスレイブが 20 台程度でマスターがスレイブからの要求を処理しきれずボトルネックとなり、スピードダウンしてしまうことが観測されている (小野 功氏とのディスカッションによる)

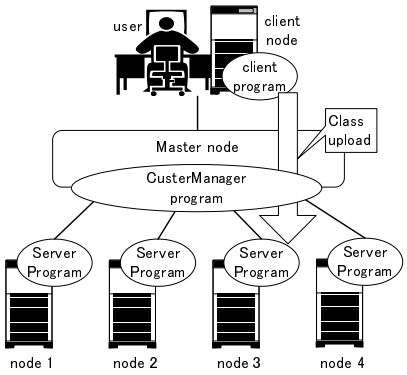

図 2 jPoP のシステム構成

通信し、ユーザが記述したクラスを各ノードにアップロードする (**図 2** を参照)。クラスローダには Sohda らによる Silf-JDSM[8] で開発された専用のクラスローダを用いる。

3.2 安 全 性

広域に分散する計算資源を安全に活用するために、異なるサイトのマスターノード間において、Globus[9] の GSI(Grid Security Infrastructure)[10] や ssh といった安全な通信をサポートする。これによって、通信路の暗号化やサイト間のセキュリティポリシーの違いといった問題を解決できる。また、ユーザコードを Grid 上で実行する場合には計算リソースをユーザコードから保護する必要がある。 jPoP の場合は各サイトに認証されているユーザのコードのみを実行可能とするため、悪意のあるユーザコードが実行される危険性はないと仮定できる。しかし、プログラムのバグから被害を与える可能性もあり注意が必要である。Java ではクラスローダ別にセキュリティマネージャを設定することができ、アプレットなどでもこれを利用してセキュリティサンドボックスを実現している。jPoP でも同様の機能を用いることで解決できる。

3.3 並列プログラミング支援

jPoP ではプログラムを書く際にユーザが記述しなければならないのは、アルゴリズムの問題依存領域部分であるデータの構造やその操作だけである。そのため、並列プログラミングで一般的に要求される、通信や同期、負荷分散について特別な記述をする必要がない。このため、ユーザが並列プログミングを意識することなく並列プログラムを書くことができる。

3.4 アルゴリズム実装支援

jPoP におけるプログラミングは特定のインターフェイス (もしくは抽象クラス) のメソッドを実装していく。これはアプレット、サーブレットなどで用いられている方法と同じである。ユーザは対象となるデータ構造とその操作方法を定義したメソッドを実装する。それぞれの最適化アルゴリズムでは、当然必要となるデータ操作が異なる。また、あるアルゴリズムのなかにも複数のアルゴリズムフレームワークがあり、それぞれ異なるデータ操作が必要となる。このため、jPoP では、個々の手法に対してそれぞれに適した抽象クラスを提供する。これにより、ユーザは最低限の定義だけでアルゴリズムを記述できる。

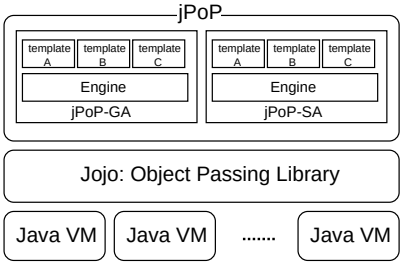

図 3 jPoP レイヤー

4. jPoP の実装

3 章で述べたように、組み合わせ最適化アプリケーションを Grid 上で実行する際に起こる並列度のスケーラビリティの問題に対し、jPoP は問題が持つ階層構造を利用した階層制御を行うことでこれを解決する。その第一段階として、jPoP の下位レイヤとして Object Passing の通信ライブラリ Jojo を設計し、ssh による実装を行った。jPoP は Jojo を用いて記述することで階層的な制御が可能になる。**図 3** に jPoP レイヤーを示す。

4.1 Jojo

Jojo は階層的な計算機構成を前提としたライブラリで、各ノード上で動くコードは自分の親ノード、子ノード、兄弟ノードと通信することができる。

通信単位は Object 一つとした。これは実装の単純さを優先したためである。より高度な通信機構はこの通信ライブラリを使用すれば容易に記述できる。パッシングの形式としては、返り値をとらない非同期送信、返り値を待つ同期送信、返り値を待つ非同期送信の 3 つを用意した。

4.1.1 実 行 環 境

通信ライブラリは jPoP の実行環境であるクラスタオブクラスタを想定して設計されている。ユーザのクライアントとなるノードをルートとする階層構造の計算機ネットワークにおいて、階層構造を意識したユーザコードを実行することで、効率的な計算を行うことを目的としている。この場合、ユーザのコードは各ノードに安全にシップされる。これによりユーザがコードをインストールする手間が省かれる。

4.1.2 プログラミングモデル

通信ライブラリのプログラミングモデルは MPI などの SPMD に近く、階層ごとの SPMD となっている。各階層の内部では同じプログラムが実行されるが、

```
abstract class Code{
  Node [] neighbors;  /*兄弟ノード*/
  Node [] lowers;     /*子ノード*/
  Node [] upper;      /*親ノード*/
  int rank;           /*兄弟の中のでの順位*/
  public abstract void init(String [] args);
  public abstract void start();
  pubiic abstract Object handle(Message meg);
}
```

図 4 Code クラス

階層間では実行されるプログラムが異なる (もちろん同じにすることもできる)。これは階層構造を前提とするシステムの記述を容易にするためである。

4.1.3 オブジェクトパッシングモデル

通信ライブラリにおける通信の単位はオブジェクトである。 MPI などとは異なり、受信はプログラムの中で明示的に行うのではなく、ハンドラを定義することで行う。ハンドラはオブジェクトの受信に伴って自動的に別スレッドで起動される。

通信同期機構としては 3 つのタイプを用意した。

非同期通信 (返り値なし) オブジェクトを送信するとただちにリターンする。

同期通信 (返り値あり) オブジェクトを送信し、それに対する返り値を待つ。

非同期通信 (返り値あり) オブジェクトを送信すると直ちにリターンするが、返り値が将来収められる Future オブジェクトを返す。Future オブジェクトに対して touch() という操作を行うことで返り値を取り出すことができる。この際、まだ返り値が収められていない (返されていない) と、返り値が収められるまでブロックする。

4.1.4 API

ユーザのプログラムには**図 4** の Code と呼ばれるクラスを実装する。各階層用に複数のコードを用意する。 init メソッドが初期化を行い、init メソッドの終了以前に start メソッドや handle メソッドが呼び出されることはない。start メソッドが本体の処理を行い、ハンドラが送信されてきたオブジェクトの処理を行う。ハンドラ (handle メソッド) を実行するスレッドは、オブジェクトを受信する毎に新たに起動されるので、ハンドラの中で長大な処理を行っても他のオブジェクトの受信に影響はでない。

4.2 jPoP の実装

jPoP は、Jojo のアプリケーションとして実現される。Jojo によって通信レイヤや実行環境が隠蔽されるので、jPoP 自身は実行環境に依存しないポータブルなシステムとなる。

jPoP は個々の解法に対してテンプレートとなる抽象クラスと、この抽象クラスを操作して解法を実行するエンジン部を提供する。ユーザはテンプレートクラスを具体的なクラスで実装することでプログラミングを行う。エンジン部は Jojo の Code クラスのサブクラスとして実現され、分散して並列に動作して解法を実行する。

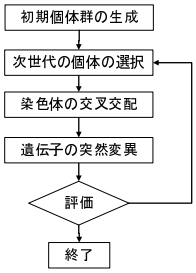

図 5 遺伝的アルゴリズムの処理手順

5. 遺伝的アルゴリズム用クラス群

jPoP のひとつとして、遺伝的アルゴリズムを分散環境で実行するための枠組である jPoP-GA を実装した。遺伝的アルゴリズムのコアとなる個体と評価環境をクラスとして定義するだけで、さまざまな分散環境で並列に個体の評価が行われる。もちろん、一つのマシンで逐次に実行する事もできる。また、GA の世代管理を行うプールに関してもユーザが独自の実装を与えることができる。さらに、個体の評価だけでなく交叉や突然変異の過程を他のマシンで行うこともできる。

5.1 遺伝的アルゴリズムの概要

2 章で述べたように GA は生物進化進化を模したシミュレーションを行うことで最適解を求めるアルゴリズムである。このアルゴリズムの一般的な処理は**図 5** のようになる。 遺伝的アルゴリズムを並列化するには島モデルによる並列化が知られているが、ここでは単純に交叉や評価の一部を他の PE で実行することを考える。この場合に留意しなければならない点は、対象となる問題によってボトルネックになる処理が異なることである。例えば、TSP を EXX[11] を用いて最適化する場合には、各個体の評価は非常に軽いが、個体の交叉にコストがかかる。一方、分子のエネルギー最小構造を求めるような場合には、交叉にはほとんどコストがかからないが、評価には非常に大きな計算量が必要になる。このため、リモートで実行する部分を下部のドライバと呼ばれる機構で自由に変更できるように設計した。ドライバによってユーザが記述する部分ではどの部分が並列実行されるかをまったく意識せずに記述することができる。

5.2 jPoP-GA の設計

ユーザが GA を記述する際には、以下の項目を記述する。

- 個体 (Individual)
- 個体を評価する環境 (Environment)
- GA のプロセス自体を制御するプール (Pool)

基本的には Pool クラスが GA の実行全体を制御

```
  /*static 変数 prop に設定されている
              SilfProperties を用いて自らを初期化*/
public static void initializeProperties();
  /*static 変数 env に設定されている
             Environment を用いて評価をし結果を返す*/
public double evaluate();
  /*引数で与えられた Individual との間に次世代の個体を
          2 個体つくり、その 2 個体からなる配列を返す*/
public Individual [] crossover(Individual);
  /*自身の突然変異個体を新たに作成して返す*/
public Individual mutate();
```

図 6 Indivdual オブジェクトをあらわすクラスに必要なメソッド

```
public interface individualFactory {
    /*prop で初期化*/
  void init(SilfProperties prop);
    /*新しい個体の作成*/
  Individual createRandomInitial();
    /*count で与えられた数からなる初期個体集合を作成*/
  Individual [] getInitialSet(int count);
}
```

図 7 IndividualFactory インターフェース

```
interface Environment extends Clonable Serializable{
     /*初期化*/
   void init(SilfProperties prop);
}
```

図 8 Environment インターフェース

するが、分散実行の制御は Pool クラスの下部にある Driver クラスで行われる。このため Pool クラスは特別な API にしたがって記述しなければならない。それぞれについて詳細に述べる。

5.2.1 個体の定義

jPoP-GA では個体をあらわす Individual オブジェクトと、個体群を生成するための Factory オブジェクトを別に記述する。

Individual オブジェクト 個体をあらわすオブジェクトは silf.jpoj.ga.Individual クラスのサブクラスとして実装する。このクラスには**図 6** のメソッドを実装しなければならない。

個体ファクトリ 個体を生成するためのオブジェクト。**図 7** の IndividualFactory インターフェースを実装しなければならない。初期個体集合の生成をつかさどる。

5.2.2 環境の定義

個体を評価する環境をあらわすオブジェクトは、**図 8** のような Environment インターフェースを実装しなければならない。

5.2.3 遺伝子プールの定義

世代交代の管理などを行う Pool オブジェクトは Pool クラスのサブクラスとして実装される。デフォルトでいくつかの Pool サブクラスを提供するので必ずしもユーザが実装する必要はないが、世代管理の手法を変更する場合はユーザが実装する事ができる。この場合には**図 9** のようなインターフェースを満たすように記述する必要がある。

図 9 にあるように Pool クラスが直接アクセスできるクラスは Individual ではなく IndividualHolder である。この IndividualHolder は基本的には Individual のラッパであり、Individual とほぼ等価なインターフェイスのメソッドを持つ。しかしこれらのメソッドには Driver クラスを呼び出すフックが埋め込まれていて、Driver クラスを経由した並列実行を実現する。

```
abstract public class Pool{
  protected SilfProperties prop;
  protected boolean minimize;
  public void init() throws SilfException{
    //dummy;
  }
    /*初期化*/
  abstract public void setInitialSet
          (IndividualHolder [] individuals);
    /*プール内の最適な個体を返す*/
  abstract public IndividualHolder getBest();
    /*プール内の全個体を返す*/
  abstract public IndividualHolder []
                                    getPopulation();
    /*次世代へ交代する*/
  abstract public void oneStep();
}
```

図 9 Pool クラス

5.3 jPoP-GA の実行

jPoP-GA の実行は以下のようにおこなわれる。

(1) Driver は IndividualFactory を用いて個体の初期集合を生成する。
(2) これをプールに与えてプールを初期化する。
(3) プロパティファイルで指定された回数だけループを実行
(4) プールの oneStep() メソッドを実行する。
(5) プールの oneStep() メソッドは、個体プールから個体を選び、交叉、評価などの処理を行う。この際に IndividualHolder を経由して制御が Driver クラス (のサブクラス) にわたり、リモートノードで交叉、評価が行われる。

5.4 定 義 例

パラメータ x に依存する関数 $x^2 + 2x + 1$ を最適化するための個体クラス**図 10** と環境クラス**図 11** の定義を示す。個体クラスの crossover メソッドで交叉交配を行い次世代の個体である新たな x の値をポワソン分布に従い生成している。生成された x は個体クラス内の evaluate メソッドで呼ばれる環境クラスの compute メソッドで評価される。並列実行ではこの環境クラスが各ノードでロードされ、Driver クラスを経由して環境クラス内のメソッドが実行される。

6. 関 連 研 究

関連する研究として、横山らによる汎用の並列最適化ソルバである PopKern[12] があげられる。PopKern は C と KLIC で実装され、共有メモリマシン上で実行される。相違点として、階層制御を行っていない点、計算機のヘテロ性に対応していない点、実装言語の制約によりポータビリティが低い点などがある。

また、Condor project[13] の MW[14] は Grid 上でのマスター・ワーカー形式のアプリケーションを容易

```
import silf.jpop.ga.*;
import silf.util.*;
import java.util.*;

public class OneDim extends Individual{
  double x;
  static Random rand = new Random();

  public OneDim(double x){
    this.x = x;
  }
  public double evaluate(){
    OneDimEnvironment env0 = (OneDimEnvironment) env;
    return env0.compute(x);
  }
              :
  public Individual [] crossover(Individual that){
      double mid = (this.x + ((OneDim)that).x) / 2;
      double dif = (this.x - mid) *
                              rand.nextGaussian();
      return new Individual[]{
  new OneDim(mid + dif),
  new OneDim(mid - dif)};
  }
              :
}
```

図 10 Individual クラスの定義例

```
import silf.jpop.ga.*;
import silf.util.*;
import java.util.*;

public class OneDimEnvironment implements
                                      Environment{
  public void init(SilfProperties prop){
    // nothing todo
  }
  double compute(double x){
    return x * x + 2 * x + 1;
  }
}
```

図 11 Environment クラスの定義例

に実行するためのソフトウェア・フレームワークであり、Condor をリソース管理に使用している。これを用いて、最適化アプリケーションを実装することは可能だが、ユーザは最適化アルゴリズムを一から実装しなければならないので負担が大きい。

7. まとめと今後の課題

本研究では、Grid 上で組合せ最適化問題を容易に解くための最適化システム jPoP の提案をした。その第一段階として Object Passing 通信ライブラリ Jojo を設計した。また、遺伝的アルゴリズムを分散環境上で実行するための枠組である jPoP-GA を設計した。現在、ssh を使用した 2 階層の Jojo が稼働している。

今後の課題として、分枝限定法と焼き鈍し法のアルゴリズムについても分散環境上で実行するための枠組を設計、実装する必要がある。また、現在 ssh で稼働している Jojo を Grid の標準的なセキュリティの枠組であるである GSI を用いて再実装する。そして、それぞれのアルゴリズムについて階層的な制御を行うことを目指す。特に分枝限定法はツリー構造の検索ツリーを状況に応じて枝刈りするので、検索ツリーの階層性を制御の階層にマップできる。これを利用することで階層的な制御が期待できる。

謝　　辞

貴重な助言、御討論をくださった東京大学の横山 大作氏、京都大学の藤沢 克樹氏、徳島大学の小野 功氏、東京工業大学の小島 政和氏ならびに小島研究室の方々、そして Ninf プロジェクトの方々に深く感謝致します。

参 考 文 献

1) 長尾智晴. 最適化アルゴリズム. 昭晃堂, 2000.
2) Ninf Project Home Page. http://ninf.apgrid.org.
3) 合田憲人, 二方克昌, 原辰次. 並列分散計算システム上での BMI 固有値問題解法. 情報処理学論ハイパフォーマンスコンピューティングシステム, Vol. 42, No. SIG12 (HPS4), pp. 132 – 141, 2001.
4) A. Takeda, K. Fujisawa, and M. Kojima. Enumeration of All Solution of a Combinational Linear Inequality System Arising from the Polyhedral Homotopy Continuation Method. *Journal of the Operations Reaserch Society of Japan*, Vol. 45, No. 1, pp. 64 – 82, 2002.
5) R.Tanese. Distributed Genetic Algorithms. In *Proc.3rd International Conference on Genetic Algorithms*, pp. 434 – 439, 1989.
6) C. Roucariro. Parallel branch and bound algorithms - an overview. In M.Cosnard et al, editor, *Parallel and Distributed Algorithms*, pp. 153 – 164, North-Holland, 1989.
7) K. Kimura and K. Taki. Time-homogeneous parallel annealing algorithm. Technical Report 673, ICOT, 1991.
8) Y. Sohda, H. Nakada, S. Matsuoka, and H. Ogawa. Implementation of a Portable Software DSM in Java. *ACMJavaGrand/ISCOPE2001 Conference*, pp. 162 – 173, Jun 2001.
9) The Globus Project. http://www.globus.org.
10) Grid Security Infrastructure. http://www.globus.org/Security/.
11) 前川, 玉置, 喜多, 西川. 遺伝アルゴリズムによる巡回セールスマン問題の一解法. 計測自動制御学会論文集, Vol. 31, No. 5, pp. 598 – 605, 1995.
12) 横山大作, 近山隆. 高度な問題領域依存チューニングを許す並列組合せ最適化ライブラリ PopKern. 情報処理学会論文誌:プログラミング, Vol. 41, No. SIG3 (PRO10), pp. 49 – 64, Mar 2001.
13) Condor Project Homepage. http://www.cs.wisc/condor/.
14) J.-P. Goux, S. Kulkarni, J. Linderoth, and M. Yorde. An Enabling Framework for Master-Worker Applications on the Computational Grid. In *Proceedings of the Ninth IEEE Symposium on High Performance Distributed Computing (HPDC9)*, pp. 43 – 50, Pittsuburgh, Pennsylvania, August 2000.