Hierarchical Parallel Simulated Annealing and Its Applications*

Shiming Xu[1], Wenguang Chen[1], Weimin Zheng[1],
Tao Wang[2], and Yimin Zhang[2]

[1] Dept. of Computer Science and Technology, Tsinghua Univ., Beijing, China
[2] Intel China Research Center, Beijing, China

Abstract. In this paper we propose a new parallelization scheme for Simulated Annealing — Hierarchical Parallel SA (HPSA). This new scheme features coarse-granularity in parallelization, directed at message-passing systems such as clusters. It combines heuristics such as adaptive clustering with SA to achieve more efficiency in local search. Through experiments with various optimization problems and comparison with some available schemes, we show that HPSA is a powerful general-purposed optimization method. It can also serve as a framework for meta-heuristics to gain broader application.

Keywords: Simulated Annealing, Parallelization, Metaheuristics, Hierarchical Clustering.

1 Introduction

Simulated Annealing(SA), firstly proposed in [7], is a randomized optimization algorithm widely applied to various combinatorial and continuous problems. Compared with other randomized algorithms, such as GA, Tabu Search, various evolutionary algorithms, it possesses a formal proof of convergence to global minima[6] under some restrictions on cooling scheduling and temperature parameters[10]. Despite this strictness, SA in practice retains the ability to avoid local minimum and to locate near-optimal solutions.

SA is computation-intensive algorithm and features sequential intrinsics; there has been much work on its parallelization [4,3,8,11,2]. With different parallel granularity, these parallel schemes are targeted at various kinds of parallel machines. Schemes of coarse granularity usually have to deal with scalability problems. We've designed a new parallel SA scheme in which processes are organized in a three-level hierarchy, addressing scalability problems effectively while achieving better coverage over the search space. Experiments show that it outperforms conventional parallel SA in either convergence speed or solution quality. This article is organized as follows: in Section 2 we will have a short overview of sequential and parallel SA and summarize related works. Detailed design and implementation of HPSA is presented afterwards in Section 3. In Section 4 we

* This project is supported by NSFC 60273007.

show HPSA outperforms available parallel SA in either speed or solution quality through various experiments. Finally We conclude that HPSA could serve as a general-purposed optimization scheme and point out our future work.

2 Sequential SA and Its Parallelization

2.1 Sequential SA

Simulated Annealing[7,1,10] is an optimization algorithm in analogy to the annealing process in metallurgy. For a formal description of SA, we give definition over these terms:

$\mathcal{S}$:	Search Space;
$\mathcal{C}ost$:	$\mathcal{S} \rightarrow \mathbb{R}$, Cost Function Defined over $\mathcal{S}$;
$\mathcal{N}$:	$\mathcal{S} \rightarrow 2^{\mathcal{S}}$, Neighborhood Function;
T :	Temperature, $T \in \mathbb{R}^+$.

SA is used to locate a solution s_m in $\mathcal{S}$ that minimize function $\mathcal{C}ost$, given the neighborhood relation $\mathcal{N}$. Usually, $\mathcal{N}$ is symmetric over $\mathcal{S}$: $\forall s \in \mathcal{S}, t \in \mathcal{S}$, $t \in \mathcal{N}(s) \rightarrow s \in \mathcal{N}(t)$. The basic idea of SA is to find an initial point in $\mathcal{S}$ and an initial temperature T_0, then conduct a random local search process within $\mathcal{S}$ under the control of T. The process carries on until T approaches zero close enough. A basic flow chart of SA is shown in Fig.1.

```
PROCEDURE Sequential_SA
BEGIN
  $s \leftarrow$ Initial Solution in $\mathcal{S}$
  $T \leftarrow$ Initial Temperature $T_0$
  DO
    DO
      $s^* \leftarrow \mathcal{N}eighbor(s)$
      $\Delta\mathcal{C} \leftarrow \mathcal{C}ost(s^*) - \mathcal{C}ost(s)$
      IF $\Delta\mathcal{C} < 0$ OR $Accept(\Delta\mathcal{C}, T)$ THEN
          $s \leftarrow s^*$
      END IF
    UNTIL Equilibrium
    $T \leftarrow$ Decrement($T$)
  UNTIL Frozen
END PROCEDURE
```

Fig. 1. Sequential SA

The outer loop of SA generally deals temperature. It starts from T_0 and terminates when T is low enough, which also terminates the algorithm. The inner loop(Metropolis Loop) which is conducted under a certain temperature, mainly deals with local search. A solution s^* in $\mathcal{N}(s)$, is generated and judged by $\mathcal{C}ost(s^*)$. If s^* is better, i.e., of lower cost, s is replaced by s^*. If it is worse, it is accepted statistically according to the Metropolis criteria[7].

2.2 Parallelization of SA

According to the classification of parallelization of Metaheuristics in [5], parallel schemes for SA fall into three categories:

- Fine granularity parallelization for inner loop
  - Functional parallelization on move evaluation
  - Data parallelization of multiple-move evaluation
- Parallelization based on search space partitioning
- Multiple concurrent runs exploring the solution space

Since Type 1 schemes [3,8,4,2,3] feature fine granularity, they fit SMP or SIMD machines. The high communication frequency between processes hampers the effectiveness of such schemes on loosely-coupled systems, such as clusters or even distributed systems. Type 2 schemes require an effective segmentation over the search space so that final output can be summarized directly basing on partial results from concurrent processes[5]. These schemes are problem-dependent, so there's much constraints applying them to general problems. In Type 3 schemes processes are organized in non-intersected subsets, which we call clusters, to conduct search process, while communication between processes follows some patterns. For further description of Type 3 schemes we define:

$\mathcal{P}$: $\{\, p_i \mid 1\leq$ i$\leq$ N$\}$, set of processes;
$\mathcal{C}$: $\{\, c_i \mid c_i \in 2^{\mathcal{P}},\ c_i \neq \emptyset,\ \bigcup_i c_i = \mathcal{P},\ c_i \bigcap c_j = \emptyset$ for $i \neq j,\ 1 \leq i, j \leq m\}$, set of clusters formed from $\mathcal{P}$.

These parallel schemes posses coarse granularity. Each process p_i in $\mathcal{P}$ initiates with a randomly chosen solution in $\mathcal{S}$ and carries on with its own chain until SA terminates. During the search process, local information is dynamically interchanged among process clusters c_j (here we assume $p_i \in c_j$) after all the processes within c_j has undergone certain steps of tempering, so that processes within c_j gain a better knowledge of the search space. Usually a solution s' is chosen or created for all the processes within c_j to carry on instead of their original solutions s_i. Process clusters could dynamically adapt during the search process.

MMC-PSA in [9] is a representative of Type 3 schemes. In MMC-PSA $\mathcal{C} \equiv \{\mathcal{P}\}$, i.e., only one constant process set exists. The replacement strategy is to replace solutions of all the processes with the best one s_best. While this replacement scheme's intuitively beneficial, currently best solution s_best may well be a local minimum. If current solutions of all processes in $\mathcal{P}$ are to be overridden, there's a possibility that processes which may potentially achieve global minimum s_m are deviated and lose adjacency to s_m. Given an extra large search space with many local minima, it is more probable for MMC-PSA to get trapped into a local minimum with fair cost, which is not our objective. Also the communication pattern of MMC-PSA does not fit large-scale systems. Especially, in asynchronous MMC-PSA, maintaining a globally accessible best solution is extremely costly in a distributed environment. HPSA is designed with these problems in notion. It's similar to MMC-PSA in that it is also based on multiple chains. Through dividing computation power over potential areas in the search space and confining most communication within process clusters, HPSA solves scalability problems faced by MMC-PSA and other similar schemes.

3 Hierarchical Parallel SA

HPSA is targeted at message passing systems, typically cluster environments. Generally HPSA can be classified as a coarse-grained, i.e., Type 3 scheme. It is similar to MMC-PSA in that it is also based on Multiple Markov Chain. Main design considerations are listed below:

– Processes include $\mathcal{P}$, a set worker processes, and a *farmer* process;
– $\mathcal{P}$ is dynamically divided into clusters: c_i's;
– *Farmer* is responsible for dynamically organizing clusters, i.e. changing $\mathcal{C}$, to achieve optimal distribution of processes, keeping:
– Processes within the same cluster have adjacent solutions, hence keeping high reachability within each cluster and minimizing the possibility of killing potential ones;
– Communication is either intra-cluster or between cluster and *farmer*.

3.1 Main Structure

Farmer process is mainly responsible for setting up and maintaining clusters. When the algorithm begins, no cluster exists. Dissociated processes, which do not belong to any cluster report to *farmer* directly. When all processes have reported to *farmer*, clustering decision is made and processes are informed of the cluster they belong. Each process is uniquely associated with a cluster, which is confined with an MPI communicator. In each cluster a head process is created to report to *farmer* at intervals about information of local search. *Farmer* decides to reshuffle clusters when a certain number of clusters have reported to have undergone great changes from their original positions. On the decision of reshuffling, *farmer* responds cluster heads with a message flag which indicates dismissal, which is broadcasted within the cluster. Processes which have received messages with dismissal flag on will become dissociated and report to *farmer* afterwards, just like when the algorithm begins. After all the clusters have been noticed of dismissal, *farmer* enters the phase same to the time when the algorithm initiates. A cluster reports to *farmer* that it's quitting when all of its processes have reported to have ended the annealing processes. When all the clusters have reported quitting, *farmer* quits, terminating the algorithm.

Communications in HPSA fall into two categories: intra-cluster communication and communication between clusters and *farmer*. Intra-cluster communication is carried out at the interval of n tempering iterations. Within a cluster the communication is synchronous, i.e., a process synchronizes with others to find out their best solution. Afterwards, processes will continue local search from this best solution. Under synchronous communication, there's no need to keep record of the globally best solution; also we are free from the overhead of exclusively accessing it.

Inter-cluster communication is fully asynchronous. Non-associated processes report to *farmer* as soon as they've reached local equilibrium under current temperature, sending out their current solutions; afterwards they wait for the cluster assignment from *farmer*. Heads of clusters report to *farmer* when local synchronization times has reached a threshold. After sending out local information to *farmer*, head processes wait until *farmer* replies. *Farmer* would either reply indicating the cluster to either carry on annealing or dismiss. On receiving dismissal messages, head process would dismiss all its fellow processes within the cluster and they will all enter non-associated state.

All the processes in HPSA are organized into a multiple-level hierarchy. When clusters are formed, it contains three levels: the highest level contains *farmer* process only; the secondary level contains all cluster head processes; the lowest layer contains ordinary working processes. When clusters are disassociated, it is a two-level structure. Under either mode, communication is controlled within directly-adjacent nodes in the hierarchy.

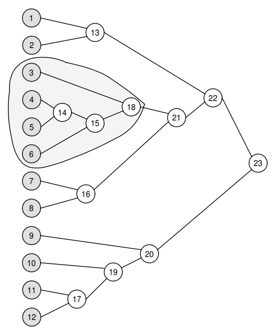
3.2 Clustering Decision in HPSA

On *farmer* we adopt Agglomerative Hierarchical Clustering to organize processes into clusters. So the process of building $\mathcal{C}$ can be divided into two steps:

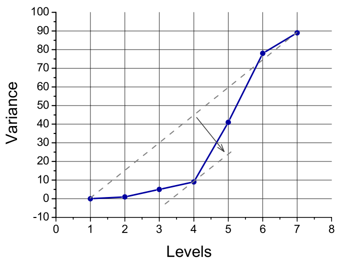
- Building Hierarchical Clustering Tree
- Forming $\mathcal{C}$

Fig.2a is an example of dendrogram of hierarchical clustering. As is shown, configurations to be clustered are labelled from 1 to 12. A full tree is formed with internal nodes labelled from 13 to 23, according to their generation time during the clustering process. While conventionally in hierarchical clustering a stop criterion is used to terminate the clustering process, such as cluster count reaches a threshold, in HPSA we decide to build the whole cluster tree since available process count is usually small and not likely to exceed several hundred and the overhead of building the whole tree is trivial. Cluster identification is specific to problems. For problems such as Protein structure prediction in [12], a quantitive threshold may be provided basing on experiences. For other problems of implicit distance measurement, threshold may be provided heuristically, for example, $1/10\ of\ Radius$.

As for those problems for which no distance threshold is available, other heuristics can be applied to identify clusters. For example, clustering decisions can be made basing on variance changes between different clustering options. For example, clustering decisions can be made basing on the variation series from leaf node up to the root in the clustering tree. See Fig.2 for an example. The variance trajectory of $Node-4$ in Fig.2a is shown in Fig.2b. Usually the variation series is non-descending. So we can detect one-step changes in variation series and put the clustering barrier between the pair of nodes with greater slope. In the previous example, all the nodes under $Node - 18$ will form a cluster.



a. Dendrogram Example



b. Trajectory of Node-4

Fig. 2. Hierarchical Clustering Example

4 Experiments

4.1 Implementation and Configurations

HPSA is implemented in MPI to support message-passing environments such as clusters. We have tested HPSA over various TSP problems.

For symmetric TSP problems, the definition of neighborhood structure and distance between solutions varies according to implementations. In HPSA we use conventional neighborhood definition for TSP[3]. With this local topology, it requires much computation to attain the distance between solutions. In HPSA we introduce a method to approximate distance between different TSP solutions: the ratio of uncovered cities by common sub-chains among all the cities of two solutions. For comparison we have implemented MMC-PSA [9,11] and MIR-PSA(Multiple Independent Run).

Experiments are carried out on an 8-node cluster, each node featuring 4-way SMP of Pentium-III Xeon 700MHz CPU and 1GB Ram. The software environment is Linux 2.4.20 and mpich-1.2.5. All nodes are connected by 100Mb/s switch. On the cluster totally 64 MPI processes are engaged in the parallel SA, including the *farmer* process.

4.2 Test Results

We have randomly picked several TSP benchmarks from TSP-LIB: *eil*101, *tsp*225, *ch*150, *kroA*100 and *kroC*100, with best solutions known. Two aspects of HPSA are evaluated: the First Hit Time(FHT) of a certain cost level and Quality of final result. Table.1 shows the test result of FHT, and cost-levels were selected as 102%,105% and 110%. Since the annealing processes are different only in the initial temperature, so the percentages of FHT in the whole annealing process is listed. The average FHT of 10 independent runs are retrieved from each test suit for HPSA, MMC-PSA and MIR-PSA. We have also tested effects of fixed scheduling on HPSA. Given a fixed initial temperature,

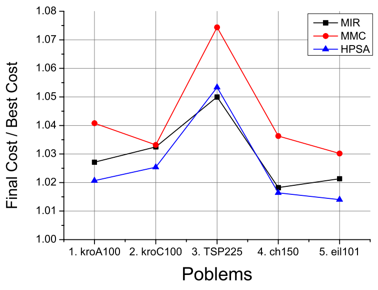


Fig. 3. Test Result II

especially one of a low value, the quenching process would take shorter time. The quality of final result generated by different parallel SA for given problems are listed in Fig.3.

Table 1. FHT Results

FHT	102%			105%			110%		
Problem	HPSA	MMC	MIR	HPSA	MMC	MIR	HPSA	MMC	MIR
ch150	83.0%	83.5%	87.0%	66.2%	64.2%	66.3%	44.0%	44.9%	34.6%
eil101	69.6%	73.2%	68.2%	57.5%	60.0%	51.9%	44.3%	48.3%	45.9%
kroA100	74.8%	77.2%	75.0%	61.2%	62.2%	64.8%	40.1%	37.0%	41%
kroC100	70.3%	70.1%	72.0%	57.7%	57.3%	61.4%	42.2%	33.1%	37.2%
tsp225	87.7%	93.3	91.0%	79.5%	59.5%	77.0%	50.0%	41.7%	47.8%

From Table.1 we can see that HPSA gains a margin over MMC-PSA and MIR-PSA if we use a lower cost level. But when cost level rises, there are more chances that any of the three may overtake the other two. Also given a fixed schedule, the quality of final result averaged by 10 runs, as is in Fig.3, shows that HPSA outperforms MMC-PSA and MIR-PSA. The fact that MIR-PSA outperforms MMC-PSA is congruent with the tuition that given a low starting temperature, MMC is more likely to kill potential processes.

The running time saved by HPSA is trivial according to our experiment results. Most of the time MMC-PSA and HPSA consume similar amount of time. Through localizing communication by assigning clusters of processes to adjacent processing units, HPSA may gain further timing-advantages over MMC-PSA.

5 Conclusion and Future Work

HPSA is a parallel SA scheme that is located between MMC-PSA and MIR-PSA. By dynamically clustering processes and manage them in a two-level hierarchy, it easily handles the scalability problem most conventional parallel SA schemes face. Through experiments we show that for TSP problems, HPSA gains advantages over MMC-PSA and MIR-PSA on the large. With further growth of distributed systems, HPSA is a more promising algorithm among parallel SA schemes.

For our future work, clustering criterion of HPSA is to be refined so that it can handle problems with speculative distance threshold is provided, which may not be accurate enough and has to be refined. Mixed clustering schemes would be more adaptive, combining both heuristics and experiential results for cluster identification. In future work we will apply HPSA to various contemporary applications, such as protein 3D structure prediction. Since HPSA can serve as an general-purposed optimization method, we will also put much emphasis on its interface design, so that we can cut down implementation efforts of applying it to other problems.

References

1. E.H.L. Aarts and J.H.M. Korst. *"Simulated Annealing and Boltzmann Machines"*. 1989.
2. A. Bevilacqua. "A Methodological Approach to Parallel Simulated Annealing on an SMP System". *J. of Parallel and Distributed Computing*, April 2002.
3. H. Chen, N.S. Flann, and D.W. Watson. "Parallel Genetic Simulated Annealing: A Massively Parallel SIMD Algorithm". *IEEE Trans. on Parallel and Distributed Systems*, 9(2), Febrary 1998.
4. R. Diekmann, R. Luling, and J. Simon. "Problem Independent Distributed Simulated Annealing and its Applications". *Proceedings of the 4th IEEE Symposium on Parallel and Distributed Processing*, 1992.
5. Fred Glover and Gary A. Konchenberger. *"Handbook of metaheuristics"*. Boston, Kluwer Academic Press, 2003.
6. V. Granville, M. Krivunek, and J. P. Rasson. "Simulated Annealing : A Proof of Convergence". *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16:652–656, June 1994.
7. S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. "Optimization by Simulated Annealing". *Science*, May 1983.
8. G. Kliewer and S. Tschöke. "A General Software Library for Parallel Simulated Annealing". *IPDPS 2000*, pages 55–61, 2000.
9. S. Lee and K.G. Lee. "Synchronous and Asynchronous Parallel Simulated Annealing with Multiple Markov Chains". *IEEE Trans. on Parallel and Distributed Systems*, 7(10):993–1008, October 1996.
10. R. Otten and L. van Ginneken. *"The Annealing Algorithm"*. Kluwer Academic Publishers, March 1989.
11. D. Janaki Ram, T. H. Sreenivas, and K. Ganapathy Subramaniam. "Parallel Simulated Annealing Algorithms". *Journal of Parallel and Distributed Annealing*, 1996.
12. K.T. Simons, C. Kooperberg, E. Huang, and D. Baker. "Assembly of Protein Tertiary Structures from Fragmens with Similar Local Sequences using Simulated Annealing and Bayesian Scoring Functions". *J. of Molecular Biology*, 1997.