A DDLM-BASED METHOD FOR SOLVING DISTRIBUTED PROBLEMS

YI TANG

Department of Mathematics
Zhongshan University,
Guangzhou, China
E-mail: tyi@guangztc.edu.cn

JIMING LIU

Department of Computer Science,
Hong Kong Baptist University,
Kowloon Tong, Hong Kong
E-mail: jiming@comp.hkbu.edu.hk

Methods based on multi-agent systems have been proposed for solving distributed problems. In this paper, we present a new method based on a distributed discrete Lagrange multiplier Method, named DDLM, for solving distributed constraint satisfaction problems. The behaviors of agents are guided by predefined DDLM rules and aggregate a global coherent behavior. Our method has been applied to some SAT benchmark problems. Experiments show that this method can yield a better performance.

1. Introduction

Many multi-agent system based methods have been proposed for solving distributed problems such as distributed CSPs (constraint satisfaction problems) [4 12]. In these methods, each agent deals with a subproblem associated with its variables and constraints. Although each agent has multiple variables and the reactive behaviors of an agent are local, the set of agents can be corporately organized and the systems can reach a solution state. Thus the individual local decisions can be aggregated to yield coherent global behavior.

Decision making problems are often associated with optimization problems. The above phenomena can be partially explained by a given optimization problem. The behavior of a local agent is as a part of the global function and the global function reflects the global behavior of multi-agent systems. Generally, the global function can be formulated as an optimal function subjected to some constraints. In a distributed multi-agent system, each agent's local decision will make the global function optimize indirectly.

Lagrange multiplier method is a common technique for optimization [2] . Traditional Lagrange method is for continuous problems [1]. Recently, a discrete Lagrange multiplier method, named DLM [9], is proposed for solving satisfiability problems. In this paper, we present a distributed discrete Lagrange multiplier method, named DDLM, for solving some distributed problems. When solving a given problem in a multi-agent system, the local reactive behavior of an agent is guided by DDLM rules and hence effects the entire behavior of problem solving. Some benchmark SAT problems are conducted to examine this method. Experimental results show that our method makes performance improvements in these problems.

The rest of the paper is organized as follows. We give the background of this paper in section 2. In section 3, we first give definitions about DDLM and then propose our search algorithm based on DDLM. In section 4, we introduce the DDLM-based method for solving distributed SAT. The experimental results and discussions are in section 5. We draw the conclusions in section 6.

2. Background

2.1. *Local search method for solving CSPs*

The methods for solving CSPs can be classified into two types: *complete*-type and *incomplete*-type. The *complete* methods can detect if a CSP has a solution and can find all solutions if there exist. But this method often require more computing costs and cannot apply to large problems. On the other hand, the *incomplete* methods can work much faster but cannot determine whether or not a CSP is feasible.

Local search method is one of the popular *incomplete* methods. It starts with a randomly initialized assignment and performs a search by changing the neighborhood of current assignment. These changes are often depended on some search strategies. The local search methods can solve many large CSPs that are beyond the capability of current *complete* methods.

2.2. *Multi-agent system for solving distributed CSPs*

In a classical environment for solving a distributed CSP, N variables are partitioned to n groups and n agents $agent_1$, $agent_2$, ..., $agent_n$ govern these groups of variables respectively. For simplicity, we assume each variable has the same domain D. The values of the variables that $agent_i$ governs compose a vector space $\mathbf{X}_i$ for each $i \in \{1, 2, ..., n\}$. The agent $agent_i$ will stay in the environment that related to the domain of $\mathbf{X}_i$ whose size is $|\mathbf{X}_i|$. When performing a local search, $agent_i$ will select a value vector $\mathbf{x}_i \in \mathbf{X}_i$ based on some predefined reactive behaviors [4]. We also say this as $agent_i$ moves to a position $\mathbf{x}_i$.

2.3. *SAT Problems*

SAT (satisfiability) Problem is an important class of CSPs. Generally, a SAT problem can be defined as follows:

Given m clauses c_1, c_2, ..., c_n on N variables x_1, x_2, ..., x_N, $x_i \in \{0, 1\}$, where each clause cl_i has the form $x_{i_1} \vee x_{i_2} \vee ... \vee x_{i_k}$ where x_{i_j} is a literal: either a variable or the negation of a variable. The whole formula in conjunctive normal form (CNF) is $cl_1 \wedge cl_2 \wedge ... \wedge cl_m$ where operator $\wedge$ denotes logical-or and $\vee$ denotes logical-and. We want to find an assignment to the variables that makes the formula true, or determine that no such assignment exists. If such an assignment is found, the system reaches a solution state.

The above SAT is a k-SAT since the number of variables in each clause is k. In this paper, we only consider the 3-SAT.

2.4. *Related Work*

It is generally reckoned that GSAT is the first SAT solver introducing the notion of local search [8]. Since then, a lot of local search methods for solving SAT have been introduced, including WSAT [5,7], Novelty [3], and Novelty+ etc. These methods are greatly efficient in finding a satisfying assignment in large SAT problems. When the system reaches a local trap, a strategy named *restart* will be applied and makes the system escape from a local minimum. In [10], another new local search method, named SDF, is proposed based on identifying the three measurable characteristics of local search behavior, i.e., *depth*, *mobility*, and *coverage*. SDF method can be regarded as one of the most efficient local search methods in solving SAT problems.

DLM (discrete Lagrange multiplier method) belongs to global search method [9,11]. It can also be used for solving SAT problems. A difference between DLM and local search is the strategy escaping from a local minimum. When the search gets stuck in a local minimum, the Lagrange multipliers can provide a power to lead the search out of the local minimum. The main algorithms include $DLM\ A_2$ [9], DLM_BASIC_SAT and $DLM\_Distance\_Penalty\_SAT$ [11], etc. Although DLM is *incomplete*, it can make better performance than many existing methods.

3. Distributed discrete Lagrange Method

In this section, we consider the following equality constrained problem in a distributed system:

$$\min_{\mathbf{x}\in\mathbf{X}_1\times\mathbf{X}_2\times\ldots\mathbf{X}_n} f(\mathbf{x})$$
$$subject\ \ to\ \ g(\mathbf{x}) = 0$$

where $\mathbf{x} = (\mathbf{x}_1,\ \mathbf{x}_2,\ ...,\ \mathbf{x}_n)^T$, $g(\mathbf{x}) = (g_1(\mathbf{x}),\ g_2(\mathbf{x}),\ ...,\ g_m(\mathbf{x}))^T$, $\mathbf{x}_i \in \mathbf{X}_i$, $i \in \{1,\ 2,\ ...,\ n\}$

The corresponding Lagrange function is as follows:

$$L(\mathbf{x},\lambda) = f(\mathbf{x}) + \sum_{j=1}^{m} \lambda_j \cdot g_j(\mathbf{x})$$

i.e.

$$L(\mathbf{x},\lambda) = f(\mathbf{x}) + \lambda^T \cdot g(\mathbf{x})$$

where $\lambda = (\lambda_1, \lambda_2, ..., \lambda_m)^T$

3.1. *Definitions*

Definition 3.1. The *interval* of two vectors $\mathbf{x} = (\mathbf{x}_1,\ \mathbf{x}_2,\ ...,\ \mathbf{x}_n)^T$ and $\mathbf{y} = (\mathbf{y}_1,\ \mathbf{y}_2,\ ...,\ \mathbf{y}_n)^T$ is k iff there exists an integer k such that $\mathbf{x}_{i_j} \neq \mathbf{y}_{i_j}$, $j \in \{1, 2, ..., k\}$ and $\mathbf{x}_{i_j} = \mathbf{y}_{i_j}$, $j \in \{k+1, k+2, ..., n\}$ where $(i_1, i_2, ..., i_n)$ is an arrangement of $(1, 2, ..., n)$ and $\mathbf{x}_i, \mathbf{y}_i \in \mathbf{X}_i$, $i \in \{1, 2, ..., n\}$. We denote it as $interval(\mathbf{x},\mathbf{y}) = 1$.

Definition 3.2. A *saddle point*$(\mathbf{x}^*, \lambda^*)$ of function $L(\mathbf{x},\lambda)$ is the point that satisfies the following:

$$L(\mathbf{x}^*,\lambda) \leq L(\mathbf{x}^*,\lambda^*) \leq L(\mathbf{x},\lambda^*)$$

for all λ sufficiently close to λ^* and for all $\mathbf{x}$ that $interval(\mathbf{x},\mathbf{x}^*) = 1$.

Definition 3.3. A neighborhood of $\mathbf{x}$ at $\mathbf{X}_i$ is a set $N_i(\mathbf{x})$ where $N_i(\mathbf{x}) = \{\mathbf{y}|interval(\mathbf{x},\mathbf{y}) = 1 \text{ and } \mathbf{x}_i \neq \mathbf{y}_i\}$.

Definition 3.4. *Distributed steepest difference gradient operator* $\triangle_{\mathbf{x}}$ is defined as follows:

$\triangle_{\mathbf{x}} L_i(\mathbf{x}, \lambda) = \mathbf{x} - \mathbf{y}$ where $L(\mathbf{y}, \lambda) = \min_{\mathbf{x}' \in N_i(\mathbf{x}) \cup \{\mathbf{x}\}} L(\mathbf{x}', \lambda)$.

3.2. *DDLM Algorithm*

The gradient descent algorithm is a class of search algorithms use the gradient of the given functions as well as the given values [2]. We formulate the main iteration for the DDLM algorithm at first. Let x^0 and λ^0 be any starting point, the DDLM iteration process is defined as follows:

$$\mathbf{x}^{k+1} = \mathbf{x}^k - \triangle_{\mathbf{x}} L_i(\mathbf{x}^k, \lambda^k) \tag{1}$$

$$\lambda^{k+1} = \lambda^k + g(\mathbf{x}^k) \tag{2}$$

where $i \in \{1, 2, .., n\}$ and the change of $\mathbf{x}^k$ is depended on the current activated agent $agent_i$, i.e., $\mathbf{x}^k$ and $\mathbf{x}^{k+1}$ are only different in the vectors in $\mathbf{X}_i$.

It is noted that the above iteration is independent with i. In each iteration, any agent can be applied. An agent $agent_i$ will determine its local behaviors by using Equation(1) to evaluate the current state. The system will update the global parameters λ by Equation(2) after yielding some local behaviors.

Theorem 3.1. *A local minimum (saddle point)* $(\mathbf{x}^*, \lambda^*)$ *of* $L(\mathbf{x}, \lambda)$ *is reached iff the iteration terminates at* $(\mathbf{x}^*, \lambda^*)$.

The proof is based on the Definition. 3.2 and Definition. 3.4 and is similar in [9].

When choosing only an agent in each iteration, we obtain the DDLM-based basic algorithm for distributed CSPs as follows:

1. **While** *not-find-solution* or *not-time-out* **do**
2. **For** all randomly selected agent a_i **do**
3. compute new-position by (1);
4. **If** *current-position* = *new-position* **then**
5. stay;
6. **Else**
7. move to new-position;
8. **End If**;
9. **If** a_i has *unsatisfied constraints* update λ by (2)
10. **If** *current state is a solution state* **Goto** 14;
11. **If** *timeout* **Goto** 15;

12. **End For**;
13. **End While**;
14. Output solution;
15. **Return**.

4. DDLM for distributed SAT

4.1. *Continuous Formulations for distributed SAT*

4.1.1. *Continuous Formulations for SAT*

We first transform a discrete SAT problem into a continuous problem. Assume x_j is a variable in a SAT where $j \in \{1, 2, ..., N\}$, the transformation $c_i(x)$ for each clause cl_i, is defined as follows:

$$c_i(\mathbf{x}) = \prod_{j=1}^{m} a_{i,j}(x_j)$$

where

$$a_{i,j}(x_j) = \begin{cases} (1 - x_j)^2 & if \;\; x_j \;\; in \;\; c_i \\ x_j^2 & if \;\; \bar{x}_j \;\; in \;\; c_i \\ 1 & otherwise \end{cases}$$

Thus if $c_i(x) = 0$ for some x, we can conclude that the constraint cl_i is satisfied.

4.1.2. *Optimization problems for distributed SAT*

Suppose an agent $agent_i$ is one of the agents in a distributed SAT solving system. Define a function $f_i(\mathbf{x})$ as:

$$f_i(\mathbf{x}) = \sum_{j=1}^{m} \varphi_{i,j}(\mathbf{x})$$

where

$$\varphi_{i,j}(\mathbf{x}) = \begin{cases} c_j(\mathbf{x}) & if \; exists \; at \; least \; a \; variable \; in \; cl_j \; governed \; by \; agent_i \\ 0 & otherwise \end{cases}$$

We can rewrite it as follows:

$$f_i(x) = \sum_{j_1} \psi_{i,j_1,1}(x) + \sum_{j_2} \psi_{i,j_2,2}(x) + \sum_{j_3} \psi_{i,j_3,3}(x)$$

where $\quad 1 \le k \le 3$
and

$$\psi_{i,j,k}(\mathbf{x}) = \begin{cases} c_j(\mathbf{x}) & if\ exists\ k\ variables\ in\ cl_j\ governed\ by\ agent_i. \\ 0 & otherwise \end{cases}$$

It can be concluded that all the constraints associated with an agent $agent_i$ will be satisfied if $f_i(\mathbf{x}) = 0$. Considering that $f_i(\mathbf{x}) = 0$ for any $\mathbf{x}$ and i, we can formulate the distributed as the following optimization problem:

$$\min \sum_{i=1}^{n} f_i(\mathbf{x})$$

$$subject\ \ to\ \ C(\mathbf{x}) = 0$$

where $\mathbf{x} = (\mathbf{x}_1,\ \mathbf{x}_2,\ ...,\ \mathbf{x}_n)^T$, $C(\mathbf{x}) = (c_1(\mathbf{x}),\ c_2(\mathbf{x}),\ ...,\ c_m(\mathbf{x}))^T$, $\mathbf{x}_i \in \mathbf{X}_i$, $i \in \{1,\ 2,\ ...,\ n\}$ and the Lagrange function is as follows:

$$L(\mathbf{x}, \lambda) = \sum_{i=1}^{n} f_i(\mathbf{x}) + \sum_{i=1}^{m} \lambda_i \cdot c_i(\mathbf{x})$$

4.2. *DDLM Algorithms for distributed SAT*

4.2.1. *DDLM-SAT-Algorithm v_1*

On the other hand, for each $agent_i$ define

$$g_i(\mathbf{x}) = \sum_{j_1}(1+\lambda_{j_1})\cdot\psi_{i,j_1,1}(\mathbf{x}) + \sum_{j_2}(2+\lambda_{j_2})\cdot\psi_{i,j_2,2}(\mathbf{x}) + \sum_{j_3}(3+\lambda_{j_3})\cdot\psi_{i,j_3,3}(\mathbf{x})$$

Let $\triangle g_i(\mathbf{x}, \lambda) = \mathbf{x} - \mathbf{y}$ where $g_i(\mathbf{y}, \lambda) = \min_{\mathbf{x}' \in N_i(\mathbf{x}) \cup \{\mathbf{x}\}} g_i(\mathbf{x}', \lambda)$. It is easy to show that $\triangle g_i(\mathbf{x}, \lambda) = \triangle_{\mathbf{x}} L_i(\mathbf{x}, \lambda)$.

The DDLM iteration for SAT can be updated as follows:

$$\mathbf{x}^{k+1} = \mathbf{x}^k - \triangle g_i(\mathbf{x}^k, \lambda^{\mathbf{k}}); \tag{3}$$

$$\lambda^{k+1} = \lambda^k + C(\mathbf{x}^k); \tag{4}$$

and we have the DDLM-SAT-Algorithm V_1 for distributed SAT by updating the preceding algorithm. According the algorithm, we find that if a SAT problem is satisfiable, there exist an assignment $\mathbf{x}$ such that $\sum_{i=1}^{n} f_i(\mathbf{x}) = 0$.

4.2.2. *DDLM-SAT-Algorithm* v_2

However, since the main iteration is based on steepest-descent and the steepest-decent like search may act as a rover ineffectively zigzag in a narrow area [2], we need other methods to alleviate the drawbacks.

When the SAT solving is processing, $agent_i$ may not move to any positions because it stays in *zero-position*, i.e. $f_i(\mathbf{x}) = \mathbf{0}$ or gets stuck in local minimum, i.e. $f_i(\mathbf{x})$ already has the minimum value but $f_i(\mathbf{x}) \neq \mathbf{0}$. We call the first as *null-movement* (zero position) and the latter as *null-movement* (local minimum). Correspondingly, when $agent_i$ moves to a new position, we call it as *agent-movement*.

Considering the Lagrange multipliers λ, it may determine the directions of agents' movements. Two simple strategies are used to update the λ. We intend to use these strategies to affect the agent *mobility* and the solving *coverage* [10].

We first define an equation for reducing the λ

$$\lambda^{k+1} = \lambda^k - \omega(\lambda^k); \tag{5}$$

where $\omega(\lambda) = (\omega(\lambda_1), \omega(\lambda_2), ..., \omega(\lambda_n))^T$ and

$$\omega(\lambda_j) = \begin{cases} 1 & if\ \lambda_j \neq 0 \\ 0 & if\ \lambda_j = 0 \end{cases}$$

And then set two counters $counter_1$ and $counter_2$ and set two threshold functions $threshold_1(\mathbf{x}, \lambda)$ and $threshold_2(\mathbf{x}, \lambda)$. The $counter_1$ records the sum of the number of *agent-movements* and *null-movements* (local minimum). The value of $threshold_1(\mathbf{x}, \lambda)$ is used to decide that if it need to update the λ by Equation(4) based on $counter_1$. When the λ is updated, we increase the $counter_2$ by 1. If $counter_2$ reaches the threshold value of $threshold_2(\mathbf{x}, \lambda)$, another procedure of updating λ by Equation(5) is fired.

The DDLM-SAT-Algorithm v_2 is as follows:

Section 1: main procedure

1. $counter_1 = 0$ and $counter_2 = 0$;
2. **While** *not-find-solution* or *not-time-out* **do**
3. **For** all randomly selected agent a_i **do**
4. compute new-position by (3);
5. **If** *current-position* = *new-position* **then**
6. stay;
7. **Else**
7. move to new-position;
9. **End If**;

10. **Call** *update* λ *procedure*
11. **If** *current state is a solution state* **Goto** 15;
12. **If** *timeout* **Goto** 16;
13. **End For**;
14. **End While**;
15. Output solution;
16. **Return**.

Section 2: *update* λ *procedure*

17. **If** a_i *encounter non null-movement(zero position)* **then**
18. $counter_1$++;
19. **If** $counter_1 \ \% \ threshold_1(\mathbf{x}, \lambda) = 0$ **then**
20. update the λ by Equation(4)
21. $counter_2$++;
22. **If** $counter_2 \ \% \ threshold_2(\mathbf{x}, \lambda) = 0$ **then**
23. update the λ by Equation(5)
24. reset $counter_1$ and $counter_2$
25. **Endif**
26. **Endif**
27. **Endif**
28. **Return**

5. Results and Discussions

5.1. *Experimental Results*

The parallel operations of distributed agents in experiments are simulated by means of sequentially dispatching agents and allowing them to make a movement based on the currently state. The agents are dispatched randomly.

We have done some experiments to examine our DDLM algorithms using the 3-SAT benchmark problems in [13]. The code was written in C and we set the two threshold functions as the agent-number, i.e., the number of agents. When runnin each instance, we initialize all x and all λ as 0.

5.1.1. *Comparison between two version DDLM-based Algorithms*

Table 1 shows the results of two versions of DDLM algorithms running 2000 times in uf100-430 test-set. This test-set includes 1000 uniform random 3-SAT instances and each instance contains 100 variables and 430 clauses. In order to avoiding much more movements of agents, we will stopped the solving when the agent-movements in DDLM v_1 exceeds 20000. The programs will be applied to a randomly selected instance with agent-number 28. We find that the v_2 performs much better than the v_1. An explanation is that the v_2 aggregates more agents' behaviors and then to decide the next moving direction, i.e., updates the multipliers λ.

Table 1. Agents movements after running 2000 times by our DDLM based SAT solver v_1 and v_2. The table means that in v_1 97 runs are stopped because the movements in these runs exceeds 20000. In other successful runs, after on average 2142 movements, the system can reach a solution state. But in v_2, the performance is improved greatly. All 2000 runs are successful, the average movements is 889. Agent-number is 28.

Algorithm ID	Success Ratio	Average Movements	Median Movement	Minimum Movement
DDLM v_1(28)	1903 / 2000	2142	1000	70
DDLM v_2(28)	2000 / 2000	889	414	28

5.1.2. *Comparison with other DLM-based Algorithms in small 3-SAT*

In Table 2, we compare the performance of four *aim*-100-2_0 problems with other two DLM-based SAT solvers. These four problems are on artificially generated random-3-SAT and from [13]. Each of them has 100 variables and 200 clauses. The result of DLM A_2 is from [9] and DLM_BASIC_SAT from [11]. In our experiments, we set agent-number as 70. The results that in solving these four small 3-SAT problems, our algorithms is better than the DLM_BASIC_SAT and exists gaps in three problems compared to the DLM A_2.

Table 2. Comparison with other DLM methods. These *aim* problem set is on artificially generated random-3-SAT and relatively small. Each problem has 100 variables and 200 clauses. The results of DLM A_2 are the average number of iteration, DLM_BASIC_SAT are are the average number of flips and DDLM v_2 are the average number of movements. In our experiments, agent-number is 70.

Problem ID	success ratio	DLM A_2	DLM_BASIC_SAT	DDLM v_2
aim-100-2_0-yes1-1	10 / 10	982	9460	2342
aim-100-2_0-yes1-2	10 / 10	1680	9473	1681
aim-100-2_0-yes1-3	10 / 10	513	5077	1574
aim-100-2_0-yes1-4	10 / 10	8510	7797	1808

5.1.3. *Comparison with other DLM-based Algorithm in large 3-SAT*

Table 3 shows another comparison with DLM_BASIC_SAT in 3-SAT problems with 200 variables. These problems are also from [13] and the results of DLM_BASIC_SAT are from [11]. We set the agent-number 56 except set 150 in aim-200-1_6-yes1-1 in our experiments. The result shows that our algorithm takes advantage in *aim*-200 problems and the DLM_BASIC_SAT takes in *uf*-200 problems.

Table 3. Comparison with other DLM methods(2). The problems are classified into two types: *aim* is on artificially generated random-3-SAT and *uf* is on uniform random-3-SAT, and relatively large. Each problem has 200 variables. The results of DLM_BASIC_SAT are the average number of flips and DDLM v_2 are the average number of movements. In our experiments, we set the number of agents with 56 except set 150 in aim-200-1_6-yes1-1.

Problem ID	success ratio	DLM_BASIC_SAT	DDLM v_2
aim-200-1_6-yes1-1	10 / 10	80877	26978
aim-200-2_0-yes1-1	10 / 10	174356	50524
aim-200-3_4-yes1-1	10 / 10	99393	12163
aim-200-6_0-yes1-1	10 / 10	894	435
uf200-01	10 / 10	11810	19008
uf200-02	10 / 10	22446	39845
uf200-03	10 / 10	1851	2507
uf200-04	10 / 10	8248	12773

5.1.4. *Comparison with other local search methods*

In Table 4 and Table 5, we compare the running results of other Local search methods in *uf*100-430 and *aim*-100 test-sets.

Our results are from the 10000 runs in *uf*100-430 and 160 runs in *aim*-100. For *uf*100-430, we applied the DDLM v_2 to a randomly selected instance in each run with agent-number 28. For *aim*-100, we run each instance 10 times with agent-number 70. The results of other methods are from [10] while the DLM algorithms is an updated version of DLM_BASIC_SAT.

The comparison results show in *uf*100-430 test-set, there exists a little gaps between our algorithm and SDF in average performance. But in *aim*-100 test-set, our algorithm performs much better than SDF and loses advantages with DLM(pars5). However our algorithm takes advantages in the two *optimal flips* indices. This means that our algorithm is surprisingly efficient in some special cases. On the other hand, it means that the variance may be larger. We need some methods to eliminate the larger variance.

Table 4. Comparison with other Local search methods(1) in $uf100$-430 test-set. The test-set has 1000 uniform random-3-SAT instances and each has 100 variables and 430 clauses. Our results is the average movements from 10000 runs with the agent-number 28.

Algorithm ID	Average Flips	Optimal Flips
WSAT(.5)	3622	2813
Novelty(.6)	2788	1273
Novelty+(.6, .01)	1581	1261
DLM(pars4)	1020	800
SDF(.00085)	864	725
DDLM v_2(28)	877	28

Table 5. Comparison with other Local search methods(2) in $aim100$ test-set. The test-set has 16 artificially generated random-3-SAT instances and each has 100 variables. Our results is the average movements from 160 runs with each instance runs 10 times.

Algorithm ID	Average Flips	Optimal Flips
WSAT(.5)	252616	252469
Novelty(.6)	251230	251002
Novelty+(.7, .01)	251587	250494
DLM(pars5)	2655	2463
SDF(.0005)	105015	102503
DDLM v_2(70)	2900	60

5.2. *Discussions*

The experiment results show the advantages of our method in solving some 3-SAT benchmark problems. Our method is based on distributed discrete Lagrange multiplier method. During the solving of a given problem, an agent will determine its local reactive behavior with Equation(1). After running some steps, various local reactive behaviors will be aggregated and force the system to tune the global parameters, i.e., the Lagrange multipliers. In SAT solving, we use Equation(4) and Equation(5) to update the Lagrange multipliers. The updating depends on two threshold functions. In our experiments, we set the two functions as a constant that is equal to the agent-number. Table 6 shows that various agent-numbers may affect the performance of solving and the optimal agent-number setting may vary in different problems.

Table 6. Effects of different agent-number

Agent-Number	success ratio	aim-100-2_0-yes1-1	aim-100-2_0-yes1-2
20	10 / 10	103201	8429
30	10 / 10	2607	5189
40	10 / 10	2694	3272
50	10 / 10	3929	2482
60	10 / 10	2935	1850
70	10 / 10	2342	2679
80	10 / 10	3713	2634
90	10 / 10	3720	1811
100	10 / 10	2455	3373

In fact, we should pay more attention to the global parameters which are the Lagrange multipliers in our solving. Not only the agent-number affects the global parameters, but also the current state of the solving system. This is why we use two functions whose variables are $\mathbf{x}$ and λ to denote the threshold of updating the Lagrange multipliers.

The use of global parameters for local search will give us a more global level to consider the problem solving [6]. However how to manage these global parameters, how to update the parameters to affect the behaviors of agents more efficiently in our special implementation, is always a problem we should address.

6. Conclusion

We present a new method for distributed problem solving based on a distributed discrete Lagrange multiplier method, named DDLM. We have examined this method in some 3-SAT benchmark problems and gain a better performance. By using Lagrange multipliers, we integrate the local reactive behaviors of agents and the global status of solving systems. Guided by this mechanism, the system can reach a solution state more efficiently.

References

1. D.P. Bertsekas, *Constrained optimization and Lagrange multiplier methods.* **Academic Press**, (1982).
2. E.K.P. Chong and S.H. Źak, *An introduction to optimization.* **John Wiley & Sons, Inc**, (1996).
3. H.Hoos, Proc. of AAAI-99, 661 (1999).

4. J. Liu, H. Jing and Y. Y. Tang, *Artificial Intelligence.* **138**, 101 (2002).
5. D.McAllester, B.Selman, and H.Kautz, Proc. of AAAI-97, 321 (1997).
6. A.Nareyek, *Constraint Programming and Large Scale Discrete Optimization, DIMACS*, **57**, 9 (2001)
7. B.Selman, H.Kautz, and B.Cohen, Proc. of AAAI-94, 337 (1994).
8. B.Selman, H.Levesque, and D.Mitchell, Proc. of AAAI-92, 440 (1992).
9. Y. Shang, B.W. Wah, *J. of global optimization.* **12**, 1 (1998).
10. D. Schuurmans and F. Southey, *Artificial Intelligence.*, **132(2)**, 121 (2001)
11. Z. Wu, Ph.D. Thesis, Dept. of Computer Science, Univ. of Illinois, (2001)
12. M. Yokoo, E.H. Durfee, T. Ishida and K. Kuwabara, *IEEE Trans. Knowledge and Data Engineering.* **10(5)**, 673 (1998).
13. http://www.intellektik.informatik.tu-darmstadt.de/SATLIB/benchm.html