A Partial Decision Scheme for Local Search Algorithms for Distributed Constraint Optimization Problems

Zhepeng Yu, Ziyu Chen*, Jingyuan He, Yancheng Deng[1]
College of Computer Science, Chongqing University, Chongqing, China
{20141402053, chenziyu, ibm_hjy}@cqu.edu.cn
[1]dyc941126@126.com

ABSTRACT

Local search algorithms are widely adopted in solving large-scale Distributed Constraint Optimization Problems (DCOPs). However, since each agent always makes its value decision based on the values of its neighbors in local search, those algorithms usually suffer from local premature convergence. More concretely, an agent cannot make a wise decision with poor values of its neighbors since its decision space is bound up with those poor values. In this paper, we propose a Partial Decision Scheme (PDS) to relax the decision space of an agent by ignoring the value of its neighbor which has the bad impact on its local benefits. The PDS comprises two partial decision processes: trigger partial decision and recursive partial decision. The former is iteratively performed by agents who cannot enhance their local benefits unilaterally to break out of potential local optima. The latter is recursively performed by neglected agents to improve global benefits. Besides, the trigger conditions along with a self-adaptive probability are introduced to control the use of PDS. The PDS can be easily applied to any local search algorithm to overcome its local premature convergence with a small additional overhead. In our theoretical analysis, we prove the feasibility and convergence of PDS. Moreover, the experimental results also demonstrate the superiority of the use of PDS in the typical local search algorithms over state-of-the-art local search algorithms.

Author Keywords: Multi-agent System; Distributed Constraint Optimization Problem; Incomplete Algorithm; Local Search Algorithm; Partial Decision

1. INTRODUCTION

With the advent of distributed artificial intelligence, multi-agent systems (MAS) [1] have become a popular way to model the complex interactions and coordination required to solve distributed problems. Distributed Constraint Optimization Problems (DCOPs) are a fundamental framework for modeling multi-agent coordination problems, widely deployed in real applications such as task scheduling [2,3], resource allocation [4], sensor networks [5,6], etc. A DCOP consists of a set of agents, each holding one or more variables. Each variable has a domain of possible value assignments. Constraints among variables assign costs to combinations of value assignments [7]. The agents must coordinate their decisions of value assignments so that the sum of all the constraints is optimized. Algorithms to solve DCOPs can be classified as being either complete [8-12] or incomplete, based on whether they can find the optimal solution or they get suboptimal solutions with small execution time. However, since DCOPs are NP-hard [7,8], complete algorithms incur exponential communication or computation with the increase of scale and complexity of problems, which limits their use in many real applications. Incomplete algorithms which require very little local computation and communication to find suboptimal solutions can be well applied to large-scale practical distributed applications. Therefore, there has been growing interest in the last few years in incomplete DCOP algorithms.

Incomplete algorithms generally include the following three categories [13]: local search algorithms, inference-based algorithms and sampling-based algorithms. Inference-based incomplete algorithms like Max-Sum [14] and its variants [15,16] allow agents to exploit the structure of a constraint graph to aggregate rewards from their neighbors but are more appropriate to acyclic DCOP graphs. Sampling-based incomplete algorithms like DUCT [17] and D-Gibbs [18] sample the search space to approximate a function as a product of statistical inference [13].

Local search algorithms are the most popular incomplete methods for DCOPs, where each agent optimizes based on its local constraints and the values received from all its neighbors, such as DSA [19], DBA [19,20] and MGM [21]. However, Local search algorithms are prone to converge to local optima. DSAN [22] tries to improve DSA by sacrificing individual benefits restricted by the neighbor values. But the algorithm is more suitable for Distributed Constraint Satisfaction Problems (DCSPs). *K*-optimality [23,24] was proposed to improve the solution of local convergence by coordinating the decision of all agents within the *K*-size coalition, such as MGM-2 [21], MGM-3 [25] and KOPT [26]. However, one of the difficulties in *K*-optimality algorithms involves the definition of *K*. Moreover, with the increase of *K*, these algorithms require greater computational effort to find a *K*-optimal solution [25]. Recently, an anytime local search (ALS) framework with some exploration heuristics [7] are presented to enhance local search algorithms. For example, DSA-PPIRA uses a periodic increase in the level of exploration and DSA-SDP employs an explorative probability with regard to the potential improvement. Besides, random restart was introduced into the ALS framework as an algorithm-independent heuristic to facilitate exploration in incomplete DCOP algorithms.

By analyzing the local search process, we find that the decision space of an agent is partitioned in terms of the values of its neighbors since its value decision heavily relies on the values of its neighbors. If always receiving the same values from its

* corresponding author.

Appears in: *Proc. of the 16th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2017),*
S. Das, E. Durfee, K. Larson, M. Winikoff (eds.)
May 8–12, 2017, São Paulo, Brazil.

neighbors, an agent will get into the same decision partitions and have no opportunity to search other promising partitions. That will lead to local premature convergence. The paper presents a partial decision scheme (PDS) to enable an agent to search the promising decision partitions by ignoring some bad value of its neighbor to enhance its local benefits, and coordinate the decision of its neglected neighbor to improve global benefits. The PDS breaks the assignment dependence on the neighbor values and help local search to escape from local optima.

The paper is organized as follows. In Section 2, we present the DCOP definition and the local search framework. Section 3 illustrates the details of our proposed scheme (PDS). Section 4 proves the feasibility and convergence of PDS. In Section 5, an experimental study is presented to evaluate PDS when combined with the typical local search algorithms in comparison with existing local search algorithms. Finally, Section 6 concludes this paper.

2. Background

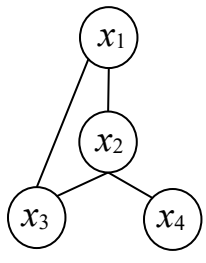
2.1 Distributed Constraint Optimization Problems

A DCOP can be represented by a tuple $\langle A, X, D, C \rangle$ such that: $A = \{a_1, a_2, \ldots, a_m\}$ is a set of agents; $X = \{x_1, x_2, \ldots, x_n\}$ is a set of variables, where each variable is assigned to an agent; $D = \{D_1, D_2, \ldots, D_n\}$ is a set of finite and discrete domains, where D_i is the domain of variable x_i; $C = \{c_1, c_2, ..., c_k\}$ is a set of constraints, where each constraint $c_i : D_{i_1} \times \ldots \times D_{i_q} \rightarrow \mathbb{R}^+$ specifies a non-negative cost for every possible value combination of a set of variables [7,8].

Given this, the goal for the agents is to find the joint variable assignment X^* such that a given global objective function is minimized. Generally, the objective function is described as the sum over C.

To facilitate understanding, this paper assumes that each agent has a single variable and constraints are binary relations. Here, the term 'agent' and 'variable' can be used interchangeably. A binary constraint is a constraint involving exactly two variables defined as $c_{ij} : D_i \times D_j \rightarrow \mathbb{R}^+$. The joint variable assignment X^* is obtained as follows:

$$X^* = \arg\min \sum_{\substack{v_i \in D_i, v_j \in D_j \\ c_{ij} \in C}} c_{ij}(x_i = v_i, x_j = v_j) \quad (1)$$



(a) constraint graph

x1 \ x2	1	2
1	4	3
2	5	1

x1 \ x3	1	2
1	1	3
2	2	4

x2 \ x3	1	2
1	2	5
2	1	4

x2 \ x4	1	2
1	2	3
2	5	1

(b) constraint matrices

Figure 1. A DCOP instance

It is common that a DCOP problem is visualized as a constraint graph where the nodes are the agents and the edges are the constraints. Figure 1 shows an example of a DCOP problem whose constraint graph and constraint matrices are shown in Figure 1(a) and Figure 1(b), respectively. The DCOP instance includes 4 variables, each of which takes a value in {1, 2}, and the cells of the constraint matrices contain the costs of the assignment in Figure 1(b).

2.2 Local Search Framework for DCOPs

The general design of local search algorithms for DCOPs is synchronous. In each round of a basic local search framework, an agent sends its value to all its neighbors in the constraint graph and receives the values of its neighbors. Then, the agent will select a value in terms of the values of its neighbors and decide whether to replace its value according to a replacement strategy. The main difference among local search algorithms is replacement strategy. For example, agents in DSA stochastically replaces their value every round if the replacements can reduce their local costs while only agents with maximal gains among their neighbors can replace their values in MGM. Here, we only present one algorithm that applies to this general framework, the Distributed Stochastic Algorithm (DSA).

Algorithm 1: Distributed Stochastic Algorithm (DSA)

For each agent x_i executes:

1. value ← *Choose_Random_Value*( )
2. send value to neighbors
3. **while** (no termination condition is met)
4. collect neighbors' value
5. select a new value which reduces the local cost most
6. Δ ← the number of the cost reduced by the new value
7. **if** (Δ > 0 and *random*( ) < *p*)
8. assign the new value
9. send value to neighbors

Figure 2. A framework of DSA

The basic idea of the DSA algorithm is simple. A sketch of DSA is presented in Figure 2. An agent x_i starts with an initial process in which x_i assigns a random value and sends the value to all its neighbors (line 1-2). Then, x_i performs a sequence of rounds until the termination condition is met. In each round (line 4-9), x_i collects the assignments of its neighbors and selects a new value to reduce the local cost most. Then, it decides, often stochastically, to keep the current value or change to the new one, if Δ > 0 (see [19] for details on the possible strategies). In this paper, we call the process in each round "*local search*".

3. Partial Decision Scheme

3.1 Motivation

Let us take Figure 1 as an example to illustrate the matter of local convergence and the idea of the proposed scheme when DSA is carried out. Assume that all agents assign as 1 after many rounds and send their value 1 to their neighbors in the next round. x_1 will receive x_2=1 and x_3=1, and try to select a new value to reduce the current local cost (line 4-6) in terms of its decision partitions pertaining to x_2=1 and x_3=1 (i.e., the white columns of the constraint matrices of x_1 shown in Figure 3). In other words, x_1 has no chance to search other partitions (i.e., the gray columns shown in Figure 3). x_1 will not change its value (line 7 will be false) since its value 1 results in the minimal local cost 5 based on the decision partition.

However, it can be observed from the constraint matrices of x_1 that the best value of x_1 is 2 with the minimal local cost 3 if x_2 and

x_3 select 2 and 1, respectively. Unfortunately, x_1 will not change its value unless x_2 and x_3 change their values. Similarly, x_2, x_3 and x_4 will not change their values upon receiving 1 from their neighbors according to the constraint matrices in Figure 1(b). At this point, DSA converges to a local minimum.

x_1 \ x_2	1	2
1	4	3
2	5	1

x_1 \ x_3	1	2
1	1	3
2	2	4

Figure 3. The decision partition of x_1 with x_2=1 and x_3=1

x_1 \ x_2	1	2
1	4	3
2	5	1

x_1 \ x_3	1	2
1	1	3
2	2	4

Figure 4. The decision partition of x_1 after ignoring x2=1

We hope to escape from a local minimum by breaking the assignment dependences among the neighbors. Our idea is to enable an agent to search the promising decision partitions by ignoring some bad value of its neighbor so as to enhance its local benefits. For example, x_1 can search more decision partitions and get the min cost 3 when ignoring x_2=1, shown in Figure 4. However, to achieve its improvement, an agent who neglects its neighbor must provide the suggested value and its gain to the neglected neighbor. The neglected neighbor will decide to accept or refuse the suggestion based on its local benefits and the received gain to ensure the improvement of global benefits. In this example, x_1 can get the local cost 3 by ignoring x_2=1 or 9 by ignoring x_3=1. Therefore, x_1 will ignore x_2=1 and change its own value to 2 under the assumption that x_2=2. Meanwhile, it will inform the suggestion (x_2=2 and gain=2) along with its new value to x_2. Based on its constraint matrices and the received gain, x_2 will get the accumulated gain by accepting x_2=2 and ignoring x_4=1. So, x_2 will change its value to 2 and inform the suggestion (x_4=2 and gain=2) along with its new value to x_4. Eventually, x_1, x_2, x_3 and x_4 take 2, 2, 1 and 2, respectively, which is the best solution to the example. On the basis of the above procedure, we propose a partial decision scheme (PDS).

3.2 Framework of Partial Decision Scheme

The PDS includes two partial decision processes, trigger partial decision and recursive partial decision. It can be easily applied to any local search algorithm (the pseudocode is shown in Figure 5).

Trigger conditions. As shown in the example of Subsection 3.1, partial decision should be performed when an agent converges to a local minimum. In other words, only if an agent cannot improve its state by performing the original local search scheme, partial decision should be triggered; otherwise, the original local search might be broken and cannot come into play when introducing partial decision. However, it is hard for an agent to identify whether the current local search process has gotten stuck in a true local minimum due to the lack of global information. Therefore, referring to quasi-local minimum (*QLM*) in [27], we introduce potential-local minimum (*PLM*) and a self-adaptive partial decision probability (*PDP*) to control the use of PDS.

Algorithm 2: PDS for a local search algorithm

For each agent x_i executes:
10. initialize *Local Search Algorithm*
11. **while** (no termination condition is met)
12. **if** receiving **suggestion_message{v^*_s, g_s}**
13. **Dispose_suggestion_message(v^*_s, g_s)**
14. calculate probability PDP_i
15. create a random number r_i ($0< r_i <1$)
16. **if** x_i is in a *PLM* and $r_i < PDP_i$
17. **Trigger_partial_decision( )**
18. **else**
19. perform *Local Search*

Trigger_partial_decision( )
20. select a neighbor x_I randomly
21. calculate value v^*_i and v^*_I by formula (8) and (9)
22. calculate the min local cost c^*_i by formula (10)
23. $g_i \leftarrow c_i - c^*_i$
24. **if** ($g_i > 0$)
25. assign value v^*_i
26. send **suggestion_message**{ v^*_I, g_i } to x_I

Dispose_suggestion_message(v^*_s, g_s)
27. $v^*_i \leftarrow v^*_s$
28. calculate the union gain g_u by formula (12)
29. **if** ($g_u > 0$)
30. assign value v^*_i
31. go to line 11
32. **else**
33. **Recursive_partial_decision**( )

Recursive_partial_decision( )
34. find x_I by formula (13)
35. calculate value v_I* by formula (14)
36 calculate g_i by formula (15)
37. **if** ($g_i > 0$)
38. assign value v^*_i
39. send **suggestion_message**{v^*_I, g_i} to x_I
40. go to line 11

Figure 5. A PDS-based local search Algorithm executed by x_i

Definition 1: An agent x_i is in a potential-local minimum (*PLM*) if x_i cannot reduce its cost unilaterally by the original local search scheme in terms of the received values from all its neighbors.

Here, *PLM* describes a state which is likely to be or evolve into a true local minimum. It plays the same role as *QLM*. However, *PLM* is a weaker condition than *QLM* since an agent detects a *PLM* only using its own current information rather than information about its neighbors. Thus, *PLM* is easier to be detected by an agent, which means the tendency of local minima can be perceived earlier. We also tested the effect of *PLM* and *QLM* on our scheme and found that the algorithms with *PLM* performed slightly better than the ones with *QLM* in terms of solution quality in the experiment.

When an agent detects that it is in a *PLM*, it triggers a partial decision with a partial decision probability (*PDP*) (line 16-17). Here, the *PDP* is formulated in terms of the quality of the current local solution and the rounds. In order to evaluate the current solution for x_i, a local cost level L_i is defined as follows:

$$L_i = \begin{cases} \dfrac{c_i - c_{\min}}{c_{\max} - c_{\min}} & , c_{\max} \neq c_{\min} \\ 0 & , c_{\max} = c_{\min} \end{cases} \quad (2)$$

$$c_i = \sum_{j \in N_i} c_{ij}(x_i = v_i, x_j = v_j), v_i \in D_i, v_j \in D_j \quad (3)$$

$$c_{\min} = \min_{v'_i \in D_i} \left\{ \sum_{j \in N_i} c_{ij} \left(x_i = v'_i, x_j = \arg\min_{v'_j \in D_j} c_{ij} \left(x_i = v'_i, x_j = v'_j \right) \right) \right\} \quad (4)$$

$$c_{\max} = \max_{v'_i \in D_i} \left\{ \sum_{j \in N_i} c_{ij} \left(x_i = v'_i, x_j = \arg\max_{v'_j \in D_j} c_{ij} \left(x_i = v'_i, x_j = v'_j \right) \right) \right\} \quad (5)$$

Here, v_i and v_j are the current value of x_i and its neighbor x_j, respectively. c_i is the local cost of the current solution for x_i. c_{min} and c_{max} are the lower and upper bounds of the local cost of x_i, respectively. N_i is an index set of all neighbors of x_i. It can be seen from formula (2) that the greater the L_i, the worse the current local solution. Additionally, x_i will record the minimal L_i computed so far. To accommodate the search process, a *PDP* for x_i is defined as the following two forms:

$$PDP_i = \sqrt{L_i} \cdot \frac{M\_round - C\_round}{M\_round} \quad (6)$$

$$PDP_i = L_i \cdot \frac{M\_round - C\_round}{M\_round} \quad (7)$$

Here, *C_round* and *M_round* represent the current and max rounds, respectively. Max rounds are the maximal synchronic iterations which are usually used as a termination condition of the local search. Initially, PDP_i is calculated according to formula (6). Once the optimal L_i retains unchanged for T rounds, PDP_i is calculated according to formula (7). In the paper, T= *M_round* /10. It can be inferred from formula (6) and (7) that x_i with greater L_i has a higher probability to trigger a partial decision and the influence of PDS will gradually wane with the increase of the rounds.

Trigger Partial Decision initiates partial decision and will be performed once the trigger conditions are met. When x_i meets *PLM* and PDP_i, it will call Trigger_partial_decision() (Line 20-26). Firstly, x_i randomly selects one of its neighbors, x_I (Line 20). Then, x_i calculates the new value v^*_i, v^*_I and the minimal local cost c^*_i by formula (8), (9) and (10), respectively (line 21-22). v^*_i and v^*_I are the values of x_i and x_I corresponding to c^*_i without regard to the received value from x_I, respectively. v_k is the received value from a neighbor x_k.

$$v^*_i = \arg\min_{v'_i \in D_i} \left\{ \sum_{\substack{k \in N_i \\ k \neq I}} c_{ik}(x_i = v'_i, x_k = v_k) + \min_{v'_I \in D_I} c_{iI}(x_i = v'_i, x_I = v'_I) \right\} \quad (8)$$

$$v^*_I = \arg\min_{v'_I \in D_I} c_{iI}(x_i = v^*_i, x_I = v'_I) \quad (9)$$

$$c^*_i = \sum_{\substack{k \in N_i \\ k \neq I}} c_{ik}(x_i = v^*_i, x_k = v_k) + c_{iI}(x_i = v^*_i, x_I = v^*_I) \quad (10)$$

$$g_i = c_i - c^*_i \quad (11)$$

Then, x_i calculates its gain g_i (line 23) by formula (11) and decides whether to change its current value to v^*_i by the following process named '*decision and replacement*' (line 24-26): If g_i is no greater than 0, which means x_i cannot improve its state when ignoring x_I, x_i will retain its current value and continue to do the next round of local search; otherwise, x_i will change to v^*_i and send a suggestion message to its neighbor x_I. The suggestion message includes the suggested value v^*_I and the gain g_i.

Dispose Suggestion Message is performed by the neglected agent to decide whether to accept the suggested value or to further perform partial decision. Upon receiving a suggestion message, x_i will execute Dispose_suggestion_message(v^*_s, g_s). Here, g_s and v^*_s is the gain and the suggested value in the suggestion message sent by x_s, respectively. x_i firstly records v^*_s as v^*_i and then calculates its union gain g_u by formula (12) (line 27-28):

$$g_u = g_s + \sum_{j \in N_i, j \neq s} (c_{ij}(x_i = v_i, x_j = v_j) - c_{ij}(x_i = v^*_i, x_j = v_j)) \quad (12)$$

If g_u is no less than 0, x_i will accept the suggestion by changing its current value to v^*_i and skip the current round of local search to avoid the conflict between local search and partial decision (line 29-31); otherwise, x_i will perform recursive partial decision (line 32-33).

Recursive Partial Decision is performed by the neglected agent to improve global benefits by ignoring one of its neighbors except its suggester. Firstly, x_i searches its decision space by ignoring the value of each neighbor except x_s to find a neighbor x_I which could minimize x_i's cost if it changed its value v_I according to formula (13) (line 34). Then, x_i computes the new value v^*_I for x_I by formula (14) and the gain g_i of all its constraints except c_{is} by formula (15) (line 35-36). Here, c_{is} has been considered in g_s in the suggestion message.

$$I = \arg\min_{\substack{j \in N_i \\ j \neq s}} \left\{ \sum_{\substack{k \in N_i \\ k \neq j, k \neq s}} c_{ik}(x_i = v^*_i, x_k = v_k) + \min_{v'_j \in D_j} c_{ij}(x_i = v^*_i, x_j = v'_j) \right\} \quad (13)$$

$$v^*_I = \arg\min_{v'_I \in D_I} c_{iI}(x_i = v^*_i, x_I = v'_I) \quad (14)$$

$$g_i = \sum_{\substack{j \in N_i \\ j \neq s}} c_{ij}(x_i = v_i, x_j = v_j) - \left(\sum_{\substack{j \in N_i \\ j \neq s, j \neq I}} c_{ij}(x_i = v^*_i, x_j = v_j) + c_{iI}(x_i = v^*_i, x_I = v^*_I) \right) \quad (15)$$

Next, x_i performs the '*decision and replacement*' process shown by **Trigger Partial Decision**. This may recursively trigger new recursive partial decision for an agent x_i receiving the suggestion message until x_i meets $g_u > 0$ or $g_i \leq 0$ (line 37-40).

It is worth noting that we use the different strategies to select a neglected neighbor, x_I. In trigger partial decision, x_I is randomly selected since the random selection requires only little computation and can avoid selecting the same neighbor repeatedly so as to enhance the diversity which is an important concern in trigger partial decision. In recursive partial decision, x_I is optimally selected since the best choice can guarantee the improvement of global benefit.

Additionally, the PDS requires a small overhead. Since the calculation for L_i requires O(1) time, and c_{min} and c_{max} can be computed in preprocessing, the overhead mainly concentrate in two parts: the extra message number and the time and space complexity caused by trigger partial decision and recursive partial decision. Actually, the number of extra messages equals to the

number of agents performing partial decision. And, each agent x_i requires $O(|D_i|\cdot(|N_i|+max_{j\in N_i}|D_j|))$ time and $O(|D_i|\cdot|N_i|)$ space in trigger partial decision, and $O(|N_i|*max_{j\in N_i}|D_j|)$ time and $O(|N_i|)$ space in recursive partial decision, which is close to the complexity of local search. Moreover, with the trigger conditions, there is only a small set of agents executing partial decision. Thus, the overall overhead is small.

4. Theoretical Analysis

We consider the implementation of the two partial decisions. Trigger partial decision is performed only if an agent meets *PLM* and *PDP*. Once an agent x_{i_0} performs trigger partial decision, there will be two cases.

case 1. Recursive partial decision is recursively performed for m times ($m \geq 0$) until the neglected agent $x_{i_{m+1}}$ accepts the suggestion from x_{i_m}, i.e., $g_u > 0$.

case 2. Recursive partial decision is recursively performed for m times ($m \geq 0$) until the neglected agent $x_{i_{m+1}}$ cannot improve its state by means of recursive partial decision, i.e., $g_{i_{m+1}} \leq 0$.

To illustrate the effects of PDS, we assume that the original local search has gotten stuck in a local minimum before using PDS, and will not increase the individual cost when using PDS.

Proposition 1 When an agent x_i performs trigger partial decision, the global cost is strictly decreasing in case 1.

proof. We begin by introducing some notations. $c^{(n)}{}_i$, $c^{(n)}_{ij}$, $C^{(n)}$ denote the local cost of x_i, the constraint cost between x_i and x_j, the global cost, respectively, at the end of the n-th round. Assume that partial decision starts with x_{i_0} ignoring x_{i_1} in the n-th round, then x_{i_1} receives the suggestion and ignores x_{i_2}, and recursive partial decision is recursively performed for m times ($m \geq 0$) until x_{i_m} ignores $x_{i_{m+1}}$ in the $n+m$-th round, and $x_{i_{m+1}}$ accepts the suggestion from x_{i_m} in the $n+m+1$-th round. Accordingly, $C^{(n-1)}$ and $C^{(n+m+1)}$ are the global costs before and after the partial decision process, respectively. We consider $C^{(n-1)} - C^{(n+m+1)}$.

$$
\begin{aligned}
&C^{(n-1)} - C^{(n+m+1)} \\
&= [\sum_{k\in N_{i_0}} c^{(n-1)}_{i_0k} + \sum_{\substack{k\in N_{i_1}\\ k\neq i_0}} c^{(n-1)}_{i_1k} + \ldots + \sum_{\substack{k\in N_{i_{m+1}}\\ k\neq i_m}} c^{(n-1)}_{i_{m+1}k} + \sum_{\substack{p\neq i_0,i_1,\ldots,i_{m+1}\\ q\neq i_0,i_1,\ldots,i_{m+1}}} c^{(n-1)}_{pq}] \\
&\quad -[\sum_{k\in N_{i_0}} c^{(n+m+1)}_{i_0k} + \sum_{\substack{k\in N_{i_1}\\ k\neq i_0}} c^{(n+m+1)}_{i_1k} + \ldots + \sum_{\substack{k\in N_{i_{m+1}}\\ k\neq i_m}} c^{(n+m+1)}_{i_{m+1}k} + \sum_{\substack{p\neq i_0,i_1,\ldots,i_{m+1}\\ q\neq i_0,i_1,\ldots,i_{m+1}}} c^{(n+m+1)}_{pq}] \qquad (16)\\
&\geq [\sum_{k\in N_{i_0}} c^{(n-1)}_{i_0k} + \sum_{\substack{k\in N_{i_1}\\ k\neq i_0}} c^{(n)}_{i_1k} + \ldots + \sum_{\substack{k\in N_{i_m}\\ k\neq i_{m-1}}} c^{(n+m-1)}_{i_mk} + \sum_{\substack{k\in N_{i_{m+1}}\\ k\neq i_m}} c^{(n+m)}_{i_{m+1}k} + \sum_{\substack{p\neq i_0,i_1,\ldots,i_{m+1}\\ q\neq i_0,i_1,\ldots,i_{m+1}}} c^{(n+m+1)}_{pq}] \\
&\quad -[\sum_{k\in N_{i_0}} c^{(n+1)}_{i_0k} + \sum_{\substack{k\in N_{i_1}\\ k\neq i_0}} c^{(n+2)}_{i_1k} + \ldots + \sum_{\substack{k\in N_{i_m}\\ k\neq i_{m-1}}} c^{(n+m+1)}_{i_mk} + \sum_{\substack{k\in N_{i_{m+1}}\\ k\neq i_m}} c^{(n+m+1)}_{i_{m+1}k} + \sum_{\substack{p\neq i_0,i_1,\ldots,i_{m+1}\\ q\neq i_0,i_1,\ldots,i_{m+1}}} c^{(n+m+1)}_{pq}] \qquad (17)\\
&= [\sum_{k\in N_{i_0}} c^{(n-1)}_{i_0k} - \sum_{k\in N_{i_0}} c^{(n+1)}_{i_0k}] + [\sum_{\substack{k\in N_{i_1}\\ k\neq i_0}} c^{(n)}_{i_1k} - \sum_{\substack{k\in N_{i_1}\\ k\neq i_0}} c^{(n+2)}_{i_1k}] + \ldots + \\
&\quad [\sum_{\substack{k\in N_{i_m}\\ k\neq i_{m-1}}} c^{(n+m-1)}_{i_mk} - \sum_{\substack{k\in N_{i_m}\\ k\neq i_{m-1}}} c^{(n+m+1)}_{i_mk}] + [\sum_{\substack{k\in N_{i_{m+1}}\\ k\neq i_m}} c^{(n+m)}_{i_{m+1}k} - \sum_{\substack{k\in N_{i_{m+1}}\\ k\neq i_m}} c^{(n+m+1)}_{i_{m+1}k}] \qquad (18)\\
&\quad +[\sum_{\substack{p\neq i_0,i_1,\ldots,i_{m+1}\\ q\neq i_0,i_1,\ldots,i_{m+1}}} c^{(n+m+1)}_{pq} - \sum_{\substack{p\neq i_0,i_1,\ldots,i_{m+1}\\ q\neq i_0,i_1,\ldots,i_{m+1}}} c^{(n+m+1)}_{pq}]
\end{aligned}
$$

$$
\begin{aligned}
&\geq g_{i_0} + g_{i_1} + \ldots + g_{i_{m-1}} + g_{i_m} + \\
&\quad \left(\sum_{k\in N_{i_{m+1}},k\neq i_m} (c^{(n+m)}_{i_{m+1}k}(x_{i_{m+1}} = v_{i_{m+1}}, x_k = v_k) - c^{(n+m+1)}_{i_{m+1}k}(x_{i_{m+1}} = v*_{i_{m+1}}, x_k = v_k))\right) \qquad (19)\\
&= \begin{cases} g_u & (m=0) \\ g_{i_0} + g_{i_1} + \ldots + g_{i_{m-1}} + g_u & (m\geq 1) \end{cases} \qquad (20)
\end{aligned}
$$

Equation (16) partitions the global cost into m+3 parts corresponding to the order of agents performing partial decision. In inequation (17), we transforms the summation by $c^{(k+1)}{}_i \geq c^{(k)}{}_i$ which is due to the fact that the agents can use local search to decrease its cost. In equation (18), we modify the summation according to the associative law of addition. In inequation (19), we transform the summation by equation (11), (15) and the fact that the gain obtained by an agent x_i between two rounds will be no less than g_i by means of the original local search. The final equation comes by equation (12). Here, g_{i_k} ($0 \leq k \leq m$-1) and g_u are greater than 0 because each neglected agent x_i assigns the suggested value $v*_i$. Thus, $C^{(n+m+1)}$ is always less than $C^{(n-1)}$ in case 1. □

Proposition 2 When an agent x_i performs trigger partial decision, the local search with PDS is equivalent to its original with local restart in case 2.

proof. Assume that partial decision starts by x_{i_0} ignoring x_{i_1} in the n-th round, then x_{i_1} receives the suggestion and ignores x_{i_2}, and recursive partial decision is recursively performed for m times ($m \geq 0$) until $x_{i_{m+1}}$ cannot improve its state by means of partial decision (i.e., $g_{i_{m+1}} \leq 0$) in the $n+m+1$-th round. Since $x_{i_{m+1}}$ does not change its value, x_{i_m} is the last agent assigning the suggested value. We consider $C^{(n-1)} - C^{(n+m)}$.

$$
\begin{aligned}
&C^{(n-1)} - C^{(n+m)} \\
&\geq [\sum_{k\in N_{i_0}} c^{(n-1)}_{i_0k} - \sum_{k\in N_{i_0}} c^{(n+1)}_{i_0k}] + [\sum_{\substack{k\in N_{i_1}\\ k\neq i_0}} c^{(n)}_{i_1k} - \sum_{\substack{k\in N_{i_1}\\ k\neq i_0}} c^{(n+2)}_{i_1k}] + \ldots + \\
&\quad [\sum_{\substack{k\in N_{i_{m-1}}\\ k\neq i_{m-2}}} c^{(n+m-2)}_{i_{m-1}k} - \sum_{\substack{k\in N_{i_{m-1}}\\ k\neq i_{m-2}}} c^{(n+m)}_{i_{m-1}k}] + [\sum_{\substack{k\in N_{i_m}\\ k\neq i_{m-1}}} c^{(n+m-1)}_{i_mk} - \sum_{\substack{k\in N_{i_m}\\ k\neq i_{m-1}}} c^{(n+m)}_{i_mk}] + 0 \qquad (21)\\
&\geq \begin{cases} \sum_{k\in N_{i_0}} c^{(n-1)}_{i_0k} - \sum_{k\in N_{i_0}} c^{(n)}_{i_0k} & (m=0) \\ g_{i_0} + g_{i_1} + \ldots + g_{i_{m-2}} + g_{i_{m-1}} + \\ \sum_{k\in N_{i_m},k\neq i_{m-1}} \left(c^{(n+m-1)}_{i_mk}(x_{i_m} = v_{i_m}, x_k = v_k) - c^{(n+m)}_{i_mk}(x_{i_m} = v*_{i_m}, x_k = v_k)\right) & (m\geq 1) \end{cases} \qquad (22)\\
&= \begin{cases} c^{(n-1)}_{i_0} - c^{(n)}_{i_0} & (m=0) \\ g_u & (m=1) \\ g_{i_0} + g_{i_1} + \ldots + g_{i_{m-2}} + g_u & (m\geq 2) \end{cases} \qquad (23)
\end{aligned}
$$

Inequation (21), (22) and equation (23) come according to (18), (19) and (20). When m = 0, x_{i_0} is the only agent changing its value after meeting *PLM*, so $c^{(n-1)}_{i_0} - c^{(n)}_{i_0}$ is less than 0. However, when $m \geq 1$, g_{i_k} ($0 \leq k \leq m$-1) is greater than 0 but g_u is less than 0 according to **Disposing suggestion message** (line 27-33). Consequently, whether $C^{(n+m)}$ is less than $C^{(n-1)}$ is unclear.

At this point, x_{i_m} has ignored $x_{i_{m+1}}$ but $x_{i_{m+1}}$ does not assign the suggested value. Thus, the current value of x_{i_m} is not the best response in terms of the values of its neighbors. So, in the next round, x_{i_m} will perform the original local search to change the current value, which could influence its neighbors, especially $x_{i_{m-1}}$. Similarly, when x_{i_m} changes its value, $x_{i_{m-1}}$ will also perform the original local search to change its value if $x_{i_{m-1}}$ holds $v^*_{i_{m-1}}$ from equation (14). The above change will happen recursively in more agents, which is equivalent to a local search with local restart. □

Proposition 3. The convergence of PDS-based algorithms will eventually depend on their originals.

proof. The convergence of PDS-based algorithms depends on two factors, the convergence of their original algorithms and the effect of PDS. It can be seen from formula (6) and (7) that *PDP* will gradually approach to zero as *C_round* grows. With the decrease of PDP, trigger partial decision is less likely to be executed (line 16). That is, the effect of PDS will be gradually decreased during the optimization process. Accordingly, more and more agents will perform the original local search (Line 18-19). Finally, PDS-based algorithms will perform just like their originals. □

5. Experimental Analysis

In order to demonstrate its effect on distributed local search, the partial decision scheme (PDS) is applied to DSA (type-C and p=0.4), DSAN, MGM and MGM2 named as PDS-DSA, PDS-DSAN, PDS-MGM and PDS-MGM2, respectively. In our experiments, we will compare these PDS-based algorithms with their originals and DSA-SDP on three different types of problems: random DCOPs, scale-free problems and graph-coloring problems. Here, DSA-SDP is reported the best ALS-DSA with exploration heuristics in [7]. We average the experimental results over 50 independently generated problems, each of which is solved by each algorithm 30 times. The compared algorithms except DSA-SDP are all implemented without the ALS framework.

We consider random DCOPs with n = 120 agents, k = 10 values in each domain, costs chosen from the range $\{1, 2,..., 100\}$, and constraint density p = 0.1 (sparse problems) or p=0.6 (dense problems) [7]. For scale-free problems, we generate instances by Barabáasi-Albert model [28] where an initial set of m_0=20 connected agents is used and in each iteration a new agent is connected to m_1=3 other agents for sparse problems or m_1=10 other agents for dense problems with a probability proportional to the number of links that the existing agents already have. Besides, the other parameter settings in scale-free networks are the same as ones in random DCOPs. For graph-coloring problems, we use n=120 agents, the number of colors=3 and the density parameter p=0.05 [7]. As in standard graph coloring problems, we set the cost of each violated constraint (two adjacent variables with the same color) to one. These problems are known to be hard Max-CSP problems, i.e., beyond the phase transition between solvable and non-solvable problems [7].

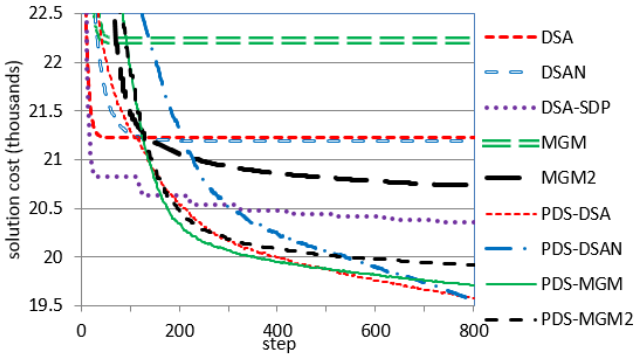


Figure 6. The cost in each step of all 9 algorithms when solving Random DCOPs (sparse problems)

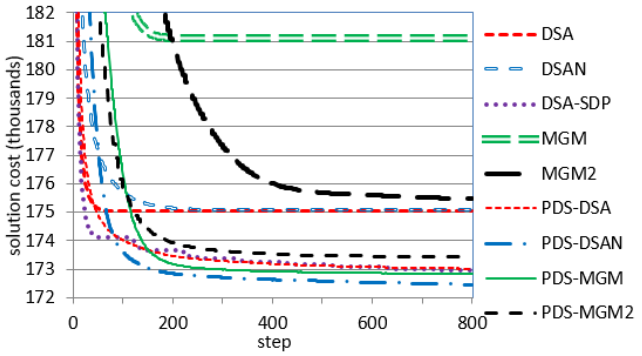


Figure 7. The cost in each step of all 9 algorithms when solving Random DCOPs (dense problems)

Figure 6 and Figure 7 show the comparison with PDS-DSA, PDS-DSAN, PDS-MGM, PDS-MGM-2, their originals and DSA-SDP on random DCOPs with p = 0.1 and p=0.6, respectively. It can be seen that the PDS-based algorithms have an obvious advantage over their originals and DSA-SDP in sparse problems. The PDS-based algorithms are superior to DSA-SDP by about 2.2%~4.0%. And the improvement of PDS-DSA over DSA is about 7.8%. PDS-DSAN improves DSAN by about 7.8%. And the improvement of PDS-MGM and PDS-MGM2 over their originals are about 11.3% and 4.0%, respectively. However, the improvements of the PDS-based algorithms over their originals are not obvious and only about 1.2%~4.5% in dense problems. Moreover, PDS-MGM2 and PDS-DSA is slightly inferior to DSA-SDP. In addition, all the PDS-based algorithms have advantages over their originals at statistically significant level of p-value $< 2.5\times10^{-10}$ in sparse problem and p-value $< 1.9\times10^{-8}$ in dense random DCOPs, respectively.

Besides, it can be found that the PDS-based algorithms have the similar curves, especially for dense problems. The reason might be that the PDS leads the local search process since it has a better search ability than the original local search schemes. Moreover, we find that PDS-MGM outperforms PDS-MGM2 while MGM2 is better than MGM. The reason is that PDS-MGM has more chances to perform partial decision in the limited steps since MGM2 and MGM requires 5 and 2 steps per round, respectively. Here, the step refers to the communication cycle between agents.

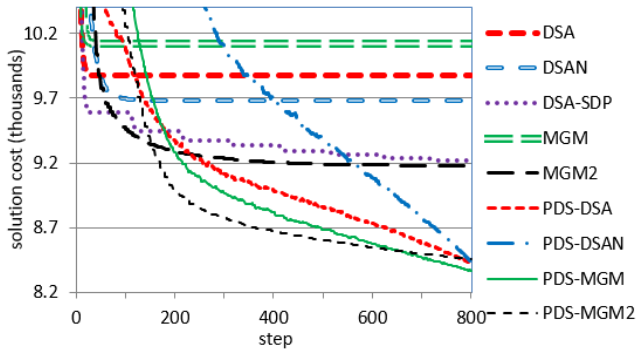


Figure 8. The cost in each step of all 9 algorithms when solving Scale-Free problems (sparse problems)

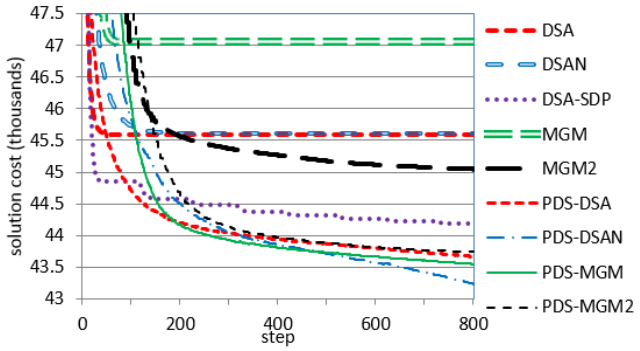


Figure 9. The cost in each step of all 9 algorithms when solving Scale-Free problems (dense problems)

Figure 8 and Figure 9 show the comparison with all nine algorithms for solving sparse and dense scale-free problems, respectively. Similar to random DCOPs, the PDS-based algorithms have great superiority over their originals in sparse problems at statistically significant level of p-value $< 9.0\times10^{-22}$. The improvement of PDS-DSA over DSA is 14.7%. And PDS-DSAN improves the DSAN by about 12.8%. PDS-MGM and PDS-MGM2 also outperform their originals by 17.4% and 7.9%, respectively. Besides, it can be seen that the PDS-based algorithms are superior to DSA-SDP by about 8.3%~9.2%. Moreover, the PDS-based algorithms have still advantages over their originals by about 3.0%~7.4% in dense problems at statistically significant level of p-value $< 3.3\times10^{-9}$.

Figure 10 show the comparison with all nine algorithms for solving graph coloring problems. We can see that DSAN and PDS-DSAN exhibit more excellent performance over the others. And, PDS-DSAN has the similar performance with DSAN, which indicates the local search scheme adopted by DSAN is more suitable for solving DCSPs. Meanwhile, it can be observed that the other PDS-based algorithms have improved their originals a lot, especially for MGM. It illustrates that the PDS is also a good decision scheme for DCSPs.

6. Conclusion and Future Work

Each agent in the local search process needs to take account of the current values of all its neighbors so as to select its value, which would prohibit it from searching some promising solutions. This

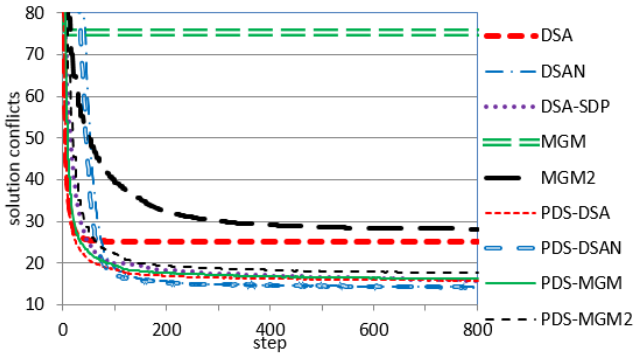


Figure 10. The cost in each step of all 9 algorithms when solving graph coloring problems

paper presents a Partial Decision Scheme to break the assignment dependence. The PDS comprises trigger partial decision and recursive partial decision processes. The former is iteratively performed by agents who meet the trigger conditions, where they neglect some bad values of their neighbors to enhance their local benefits and provide the suggested values for the neglected neighbors. The latter is recursively performed by neglected agents, where they ignore the values of their neighbors except their suggesters to improve global benefits. The partial decision scheme can be applied to any local search algorithm and the experimental results verify its advantage on benchmark problems.

In the future, we will probe into new solution evaluation and control mechanism to enhance the convergence speed and accuracy of PDS-based local search algorithms. In addition, we will also try to extend the partial decision scheme by ignoring the values of multiple neighbors.

7. REFERENCES

[1] Pujol-Gonzalez, M. 2011. Multi-agent coordination: Dcops and beyond. In: *Proceedings International Joint Conference on Artificial Intelligence* (IJCAI) (Vol. 22, No. 3, p. 2838).

[2] Enembreck, F., Barthès, J. P. A. 2012. Distributed constraint optimization with MULBS: A case study on collaborative meeting scheduling. *Journal of Network and Computer Applications*, 35(1), 164-175.

[3] Sultanik, E., Modi, P. J., Regli, W. C. 2007. On Modeling Multiagent Task Scheduling as a Distributed Constraint Optimization Problem. In: *Proceedings International Joint Conference on Artificial Intelligence* (IJCAI). pp. 1531-1536.

[4] Cheng, S., Raja, A., Xie, J. 2014. Dynamic multi-agent load balancing using distributed constraint optimization techniques. *Web Intelligence and Agent Systems: An International Journal*, 12(2), 111-138.

[5] Farinelli, A., Rogers, A., Jennings, N. R. 2014. Agent-based decentralised coordination for sensor networks using the max-sum algorithm. In: *Autonomous agents and multi-agent systems*, 28(3), 337-380.

[6] Muldoon, C., O'Hare, G. M., O'Grady, M. J., Tynan, R., Trigoni, N. 2013. Distributed constraint optimization for resource limited sensor networks. In: *Science of Computer Programming*, 78(5), 583-593.

[7] Zivan, R., Okamoto, S., Peled, H. 2014. Explorative anytime local search for distributed constraint optimization. *Artificial Intelligence*, 212, 1-26.

[8] Modi, P. J., Shen, W.-M., M. Tambe, and M. Yokoo. 2005. Adopt: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence*, 161(1):149-180.

[9] Petcu, A., Faltings, B. 2005. A scalable method for multiagent constraint optimization. In: *Proceedings of the International Joint Conference on Artificial Intelligence* (IJCAI), pp. 266–271.

[10] Gershman, A., Meisels, A., Zivan, R. 2009. Asynchronous Forward-Bounding for distributed COPs, *Journal of Artificial Intelligence Research*, 34, 61–88.

[11] Hirayama, K., Yokoo, M. 1997. Distributed partial constraint satisfaction problem. In: *Proceedings of the International Conference on Principles and Practice of Constraint Programming* (CP), pp. 222–236.

[12] Mailler, R., Lesser, V. 2004. Solving distributed constraint optimization problems using cooperative mediation. In: *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems* (AAMAS), pp, 438-445.

[13] Fioretto, F., Pontelli, E., Yeoh, W. 2016. Distributed Constraint Optimization Problems and Applications: A Survey. *arXiv preprint arXiv*:1602.06347.

[14] Farinelli, A., Rogers, A., Petcu, A., Jennings, N. 2008. Decentralised coordination of low-power embedded devices using the Max-Sum algorithm. In: *Proceedings of the International Conference on Autonomous Agents and Multi-agent Systems* (AAMAS), pp. 639–646.

[15] Rogers, A., Farinelli, A., Stranders, R., Jennings, N. 2011. Bounded approximate decentralised coordination via the max-sum algorithm. *Artificial Intelligence* 175 (2), 730–759.

[16] Rollon, E., Larrosa, J. 2012. Improved Bounded Max-Sum for distributed constraint optimization. In: *Proceedings of the International Conference on Principles and Practice of Constraint Programming* (CP), pp. 624–632.

[17] Ottens, B., Dimitrakakis, C., Faltings, B. 2012. DUCT: An upper confidence bound approach to distributed constraint optimization problems. In: *Proceedings of the National Conference on Artificial Intelligence*. (Vol. 1, No. EPFL-CONF-197504, pp. 528-534).

[18] Nguyen, D. T., Yeoh, W., Lau, H. C. 2013. Distributed Gibbs: A memory-bounded sampling-based DCOP algorithm. In: *Proceedings of the International Conference on Autonomous Agents and Multi-agent Systems* (AAMAS), pp. 167-174.

[19] Zhang, W., Wang, G., Xing, Z., Wittenberg, L. 2005. Distributed stochastic search and distributed breakout: Properties, comparison and applications to constraint optimization problems in sensor networks. *Artificial Intelligence*, 161 (1–2) 55–87.

[20] Hirayama, K., Yokoo, M. 2005. The distributed breakout algorithms. *Artificial Intelligence*, 161(1), 89-115.

[21] Maheswaran, R., Pearce, J., Tambe, M. 2004. Distributed algorithms for DCOP: A graphical game-based approach. In: *Proceedings of the International Conference on Parallel and Distributed Computing Systems* (PDCS), pp. 432–439.

[22] Arshad, M., Silaghi, M. C. 2004. Distributed simulated annealing. In: *Distributed Constraint Problem Solving and Reasoning in Multi-Agent Systems*, 112.

[23] Pearce, J., Tambe, M. 2007. Quality guarantees on k-optimal solutions for distributed constraint optimization problems. In: *Proceedings of the International Joint Conference on Artificial Intelligence* (IJCAI), pp. 1446–1451.

[24] Vinyals, M., Shieh, E., Cerquides, J., Rodriguez-Aguilar, J., Yin, Z., Tambe, M., Bowring, E. 2011. Quality guarantees for region optimal DCOP algorithms. In: *Proceedings of the International Conference on Autonomous Agents and Multi-agent Systems* (AAMAS), pp. 133–140.

[25] Leite, A. R., Enembreck, F., Barthès, J. P. A. 2014. Distributed constraint optimization problems: Review and Perspectives. *Expert Systems with Applications*, 41(11), 5139-5157.

[26] Katagishi, H, Pearce, J. P. 2007. KOPT: Distributed DCOP Algorithm for Arbitrary K-optima with Monotonically Increasing Utility. DCR-07.

[27] Okamoto, S., Zivan, R., Nahon, A., et al. 2016. Distributed Breakout: Beyond Satisfaction. In: *Proceedings of the International Joint Conference on Artificial Intelligence* (IJCAI), 447-453.

[28] Barabáasi, A.-L., Albert, R. 1999. Emergence of scaling in random networks. *Science*, 286(5439):509-512.