A Parallel Heuristic Method for Optimizing a Real Life Problem (Agricultural Land Investment Problem)

Sagvan Ali Saleh
College of Agriculture, University of Duhok, Duhok, Kurdistan region, Iraq

ABSTRACT

This paper proposed a parallel method for solving the Agricultural Land Investment Problem (ALIP), the problem that has an important impact on the agriculture issues. The author is first represent mathematically the problem by introducing a mathematical programming model. Then, a parallel method is proposed for optimizing the problem. The proposed method based on principles of parallel computing and neighborhood search methods. Neighborhood search techniques explore a series of solutions spaces with the aim of finding the best one. This is exploited in parallel computing, where several search processes are performed simultaneously. The parallel computing is designed using Message Passing Interface (MPI) which allows to build a flexible parallel program that can be executed in multicore and/or distributed environment. The method is competitive since it is able to solve a real life problem and yield high quality results in a fast solution runtime.

KEYWORDS : parallel processing, neighborhood search, agriculture.

1. INTRODUCTION

This paper addressed an issue of agricultural problems, namely the Agricultural Land Investment Problem (abbreviated to ALIP). The problem faced by an investor, who needs to invest a large agricultural land. There are various variants of plants that can be cultivated with a limited finance and duration. Each plant has a cost to be raised. On the other hand, each of them has a profit. These profits varies according to plants. Some plants have great profits but they cost a lot until they can bear fruit and vice versa. The objective of the problem is to maximize the profit of investment of Agriculture. As it is clear that, the ALIP is NP-hard. In order to simplify the treating of the problem, the ALIP can be simulated as a well-known combinatorial optimization problem that is the knapsack problem (abbreviated to KP). In fact, there are a wide variety of practical situations that can be simulated as the KP in various domains, including, computer sciences (Kellerer *et al.*, 2014). Given a set I on n plants and financed with a limited capacity c, where each plant i is characterized by a profit P_i and a cost w_i, the objective of the problem is to select a subset of plants so that the sum of the selected plants' profits is maximized without exceeding the limited financed capacity c. The mathematical programming model of the ALIP can be stated as follows :

$$Max: \quad f(x) = \sum_{i=1}^{n} p_i\, x_i \qquad \ldots\ldots\ldots\ldots\ldots (1)$$

$$s.t. \qquad \sum_{i=1}^{n} w_i\, x_i \le c \qquad \ldots\ldots\ldots\ldots\ldots (2)$$

$$x_i \in \{0,1\} \quad \forall i \in I = \{1, \ldots, n\} \qquad \ldots\ldots\ldots\ldots\ldots (3)$$

The decision variable x_i, $\forall\, i \in I$, is equal to 1 ($i.e.\ x_i = 1$) if a plant i is selected and cultivated in the land (included in the solution), otherwise $x_i = 0$ (out of the solution). In this integer linear programming, there are three equations. The first equation (1) is the objective function where the goal is to maximize the value of the total profit of plants cultivated in the land under two constraint. The first constraint (equation 2) is the capacity constraint, ensuring that the costs' sum of the selected plants does not exceed the limited finance. Meanwhile, the second constraint (equation 3) is imposed on plants that are to be selected or not in the solution (it is not allowed to select a fractional plant) (Martello & Paolo, 1990). In order to avoid trivial cases, it is assumed that: all input data $c, w_i, p_i, \forall\, i \in I$, are positive integers, and $\sum_{i \in I} w_i > c$ (Hifi & Otmani, 2012). The rest of the paper is organized as follows. Section two reviews some related works. Section three discusses the principle of the proposed approach for optimizing the ALIP. In section four, the performance of the proposed approach is evaluates on a number of instances, and analyzes the obtained results. Finally,

Academic Journal of Nawroz University
(AJNU) Volume 7, No 4 (2018).
Regular research paper : Published 21 December 2018
Corresponding author's e-mail : sagvan.saleh@uod.ac

doi : 10.25007/ajnu.v7n4a286

section five summarizes the contribution of the paper.

2. Related Works

The ALIP is a real life problem that can be simulated as the KP, as mention in the previous section (i.e. Section 1). Accordingly, the solution procedures for the problems belonging to the knapsack family are also suitable for the ALIP (for more details, see Kellerer *et al.*, 2014). However, the solution procedures, available in the literature, are either exact or approximate methods. Horowitz and Sahni (Horowitz & Sahni, 1974) proposed a branch and bound algorithm, which is based on a depth first enumeration. The upper bounds is derived from the principles of continuous relaxation of the currently induced sub-problem. On the other hand, in order to solve large-sized instances, Balas and Zemel (Balas & Zemel, 1980) proposed a method focused on the enumeration on the most interesting ones. This subset of items, known as the core of the problem, is then solved either by approximate techniques or by using exact solution procedure such as branch-and-bound methods. Yamada *et al.* (Yamada *et al.*, 2002), proposed exact and approximate algorithms for solving a variation of the knapsack problem that is disjunctively constrained knapsack problem. The heuristic solution procedure is used in order to generate approximate feasible solutions. Then, these solutions are improved using local search method. The exact solution method is based on the principles of implicit enumeration search which starts its search from the initial solution obtained from the approximate part. Hifi *et al.* (Hifi *et al.*, 2014) proposed a parallel method based on the principles of large neighborhood search method for solving the disjunctively constrained knapsack problem. The method is designed using message passing interface. In this paper, we proposed an approximate solution procedure based on principles of parallel computing and neighborhood search methods. The neighborhood search techniques are exploited in parallel computing, where several search processes are performed simultaneously.

3. Parallel heuristic based on neighborhood search

In this section, we discuss the principle of the proposed parallel heuristic method which is based on the principles of neighborhood search for optimizing the ALIP problem. Neighborhood search is a wide class of improvement techniques that can be used for developing effective algorithms to approximate large-size instances of various combinatorial optimization problems (Aarts & Lenstra, 2003). On the other hand, parallel computing have already been proved as an effective solution procedure in which several calculations can be performed simultaneously. The idea is that huge data can be divided into smallest parts, which are then processed and treated concurrently (Pacheco, 2011). In this paper, the neighborhood search techniques are exploited in parallel computing. Accordingly, a parallel heuristic method is presented for solving the ALIP problem. On the one hand, neighborhood search techniques explore a series of solutions spaces with the aim of finding the better. On the other hand, several search processes are performed simultaneously and this dramatically accelerates the computing processes and yields high quality solutions within acceptable solution running time.

3.1 Parallel computation and MPI

It has become more interesting than before to design algorithms that can be implemented effectively in a multiprocessor/multicore environment. Such that, we have numbers of processing units which cooperate for solving a specific problem. That is, the parallel computing. This will lead to a high progress in the solution procedures of the considered problem (Pacheco, 2011). In this work, we use Message Passing Interface (noted as MPI) to design a parallel heuristic method that can be executed in multicore and/or distributed environment. MPI addresses primarily the message-passing parallel programming model, in which data is moved from the address space of one processor to that of another processor through cooperative operations on each processor. The main advantage of MPI is the ease of use. In addition, it enables us to build an efficient, and flexible message passing model of parallel programming (Wittwer, 2006).

3.2 Parallel Computation Environment

The proposed parallel computation model provides a multiprocessor environment. In such environment, there are multiprocessors on a single multicore personal computer machine. This is a cheap computing power that has been exploited in our parallel programming model. In fact, the tackled problem is divided into sub-problems, then each one of them is treated by one core of the multicore personal computer. In other words, we have multi-processing environment, where each process is associated with a unique core. All processes are arranged and ordered within a virtual topology known as Communicator. The virtual topology means that, there may be no relation between the physical structures of the process topology. In contrast, it describes a virtual mapping ordering of the processes. The mapping of processes in the MPI communicator is dependent upon the algorithm implementation and the problem to be solved. The communicator composed of a set of processes, where each one of them has a unique integer rank. Rank value start at zero to $N-1$, where N is the total number of processes in the communicator. The whole processes in the communicator may communicate with each other using their ranks either by collective or point to point communications operations. A sender process may send a message to other process by providing the receiver ranks and a

doi : 10.25007/ajnu.v7n4a286

unique tag to identify the message. The receiver process can then post a receive for a message by providing the sender rank and the same tag to identify the message (for some cases it can simply ignore the tags). Accordingly, the receiver process handles the data correctly (Pacheco, 2011).

3.3 Parallel heuristic Implementation

The solution method proposed for optimizing the ALIP problem focus on the computation of high quality solutions with the acceptable computational effort. Thus, the proposed method is an approximative methods, due to the nature of the problem which is NP-hard problem. Algorithm 1 describes the main steps of the proposed parallel heuristic method. The principle is that, several neighborhood search processes are performed concurrently by the help of MPI parallel programming model. Each core in the used computer is associated with one process, where a random neighborhood search is considered as an efficient search process. In other words, each core perform a neighborhood search process according to its parameters, then the best results obtained from the whole processes are recorded. Herein, an adaptive neighborhood search method is considered, which consists of two main stages: the first stage yields a starting feasible solution by using a greedy procedure. The second stage tries to improve the quality of the starting feasible solution by using local search and a diversification strategy. The diversification strategy is used in order to escape from a series of local optimum solution and explore diversify search spaces as shown in algorithm 1.

Algorithm 1: Parallel Heuristic method for the ALIP

Require: S_{ALIP}, a starting solution of P_{ALIP}

Ensure: S^*_{ALIP}, the best local optimum solution of P_{ALIP}

1. Initialize S_{ALIP}, the starting solution, using a greedy procedure.
2. Initialize n processors and make the processor;
3. On each processor i $(i = 0, 1, \dots, n-1)$, apply a random neighborhood search:
3.1 While the iteration limit is not achieved do
3.1.1 Use the starting solution as the initial solution
3.1.2 Apply a destroying strategy to yields a reduced sub-problem.
3.1.3 Apply a local search procedure to improve the reduced sub-problem.
3.1.4 Update the best solution at hand
3.2: End While
4. Update S^*_{ALIP} the best local solution obtained from all processes
5. Return S^*_{ALIP}.

Algorithm 1 describes the principles of the proposed parallel method for optimizing the ALIP. Step 1, a starting solution is initialize using a simple greedy procedure, in which items are selected randomly to be included in the solution. In this work, we use n equals to 4. That is mean that, we use only four cores in the multicore computer. Step 2: n processors are initialize, where the neighborhood search process will be applied. Step 3: apply a diversification neighborhood search strategy, which includes two steps: a degrading strategy (Step 3.1.2) and a re-optimizing strategy (Step 3.1.3). The degrading strategy destroy the solution at hand by removing some elements randomly and produced a reduced sub-problem. The re-optimizing strategy tries to enhance the reduced sub-problem by applying a local search procedure. Step 4 update the best solution achieved by all processes. Finally, the best solution obtained so far is returned.

4. Computational Results

This section investigates the effectiveness of the proposed parallel heuristic neighborhood search method (abbreviated to PHNS) on instances generated by the author. The instances consists of 200 variants of plants, and a limited finance $c = 1200\$$. The results obtained by the proposed PHNS are compared to those results obtained by the sequential version of the proposed method. The proposed PHNS was coded in C++ and MPI library and tested on core i5 2.5 GHz.

There are two parameters should be taken into account with respect to neighborhood search: the percentage of the removed elements regarding the reduced problem (noted as $\propto$) and the stopping criteria. Both of them are important in developing an efficient algorithm. For the first step in the computational results, we tune our algorithm as following: we initialize four processes in four different cores in the used personal computer. Each process handle a variant of $\propto$. The first process handle $\propto = 10$, the second $\propto = 20$, the third $\propto = 30$, and the fourth process handle $\propto = 40$. In the other hand, for all processes the stopping criteria has been fixed to 2000 iteration. In other words, neighborhood search method iterates 2000 times before it gives its better solution. Table 1 displays the variation of the average solutions values and the solution time achieved by PHNS over the treated instances.

doi : 10.25007/ajnu.v7n4a286

Table (1) : The effect of the variable ∝ on the solution quality and time

No. of trials	**α=10**		**α=20**		**α=30**		**α=40**	
	t (s)	Av. Val	t (s)	Av. Val	t (s)	Av. Val	t (s)	Av. Val
1st	0.109	9318	0.124	8322	0.126	7854	0.158	7798
2nd	0.106	8022	0.094	8920	0.124	8208	0.156	7772
3rd	0.124	9044	0.124	8796	0.14	8354	0.14	8020
4th	0.11	8522	0.095	9244	0.126	7828	0.126	7684
5th	0.109	8654	0.094	8516	0.156	8154	0.171	7676
Av. Tot	0.1116	8712	0.1062	8759.6	0.1344	8079.6	0.1502	7790

Table 1 illustrates the effectiveness of the proposed parallel heuristic on the tested instances. The first column (No. of trials) shows the number of trials, where we did 5 trials. Column 2 ($\alpha = 10$) illustrates the average time and the average solutions obtained with different trails. As it is clear from the table, the best solution achieved is when $\alpha = 20$. So, we tune our algorithm with this value for the next step regarding our computational results. For the next step in the computational results, we use the same number of processes, (i.e. four processes in four different cores). We tune all processes with $\alpha = 20$. At the same time, we extend the stopping criteria to 400000 iteration. This extension in the number of iteration, normally, improves the solution quality at the expense of required runtime, as shown in Table 2.

Table (2) : Performance of PHNS with compare with a sequential algorithm

No. of	**Seq. Algo.**		**PHNS: α=20**	
	t (s)	**Value**	**t (s)**	**Value**
1st	5.184	9192	1.435	9316
2nd	5.22	9240	1.513	9200
3rd	5.193	9170	1.466	9352
4th	5.166	9260	1.482	9242
5th	5.122	9428	1.45	9278
Av.	5.177	9258	1.469	9277.6

Table 2 illustrates the performance of the PHNS. Column 1 (No. of trials) shows the number of trails. Column 2 (Seq. Algo.) shows the solutions obtained and the time required for a sequential form of the neighborhood search algorithm. Column 3 (PHNS: α=20) illustrates the solutions and time required for PHNS. One can observe that, the sequential algorithm required about 5.177 s in order to yields its solutions values while, the parallel version, yields its solutions values within average runtime equals to 1.469 s. That's mean, the parallel algorithm is faster than the sequential one, about $\frac{5.177}{1.469} = 3.52$ times faster than the sequential one. At the same time, the performance of the parallel algorithm and the sequential one are nearly the same. That is because, the both used the same neighborhood search techniques. For the third and last step in the computational results, we compare the PHNS, with the greedy algorithm that produce the initial solution, as illustrated in Table 3.

Table (3) : Performance of PHNS with compare with a greedy algorithm

	Greedy algorithm		PHNS: α=20	
	t (s)	Val.	t (s)	Val.
Av. Val.	0.031	3322	1.469	9277.6

As shown in Table 3, the average solution value obtained by PHNS is better than the initial solution obtained by the greedy algorithm. Indeed, PHNS is able to realize an average value about 9277.6 in 1.469 second, whereas Greedy algorithm realizes 3322 in 0.031. In spite of PHNS required more solution time than the greedy algorithm, but the solution quality is better.

5. Conclusion:

In this paper, a parallel heuristic method was proposed for approximately solving a real life problem arises in agriculture, agricultural land investment problem. The problem was first modeled as a combinatorial optimization problem, the knapsack problem. In addition, a mathematical programming model was introduced to represent it. Then, a parallel heuristic method is proposed. The proposed method based on the principles of parallel computing and neighborhood search techniques. The neighborhood method is based upon destroying and re-optimizing strategies. Destroying strategy involves remove α% of items from the solution at hand, whereas re-optimizing strategy involves re-optimize the reduced problem. This is exploited in parallel computing, where several search

doi : 10.25007/ajnu.v7n4a286

processes are performed simultaneously. The Message Passing Interface was used to build the parallel program in a multicore environment. The work realized here highlighted the effectiveness of integrating neighborhood search techniques in parallel processing. Computational results showed that, the proposed method performed better than a greedy algorithm and a sequential version of neighborhood search. The proposed method yields high quality solutions in short solution time.

References:

1. Aarts , E., & Lenstra, J. (2003), Local search in combinatorial optimization. New Jersey: Princeton University Press.
2. Balas, E., & Zemel, E. (1980), An algorithm for large zero-one knapsack problems. Operations Research 28, 1130-1154.
3. Hifi, M., & Otmani, N. (2012), An algorithm for the disjunctively constrained knapsack problem. International Journal of Operational Research , 13(1): 22-43.
4. Hifi, M., Negre, S., Saadi, T., Saleh, S., & Wu, L. (2014), A Parallel Large Neighborhood Search-Based Heuristic for the Disjunctively Constrained Knapsack Problem. IEEE International Parallel & Distributed Processing Symposium Workshops. Phoenix, AZ, USA: IEEE.
5. Horowitz , E., & Sahni, S. (1974), Computing partitions with applications to the knapsack problem. Journal of the ACM (JACM), 21 (2): 277–292.
6. Kellerer, H., Pferschy, U., & Pisinger, D. (2014), Knapsack Problems. New Yourk: Springer-Verlag Berlin Heidelberg.
7. Martello , S., & Paolo, T. (1990), Knapsack problems. New York: Wiley .
8. Pacheco, P. (2011), An Introduction to Parallel Programming. USA: Morgan Kaufmann Publishers.
9. Wittwer, T. (2006), An Introduction to Parallel Programming. The Netherlands: VSSD.
10. Yamada, T., Kataoka, S., & Watanabe., K. (2002), Heuristic and exact algorithms for the disjunctively constrained knapsack problem. Information Processing Society of Japan Journal, 43: 2864-2870.

doi : 10.25007/ajnu.v7n4a286