Applying Genetic Algorithms and Other Heuristic Methods to Handle PC Configuration Problems

Vincent Tam and K.T. Ma

{vtam, makengte}@comp.nus.edu.sg
Department of Computer Science
School of Computing, National University of Singapore
Lower Kent Ridge Road, Singapore 119260.

Abstract. Singapore is developing very fast as an Information Technology (IT) hub in which many people are keen to configure and build their own personal computers (PC). Like many real-life configuration problems, a well-designed PC configuration often represents a challenge in which given the wide diversity of hardware components, the ever-changing PC technology and the limited compatibility between some hardware components, we are interested to obtain an (sub-)optimal configuration for each specific usage restricted to a budget limit and other preferred criteria. In this paper, we formally defined these PC configuration problems as discrete optimization problems. Then we proposed a systematic and flexible framework in which we can integrate any heuristic search method for solving these difficult real-world discrete optimization problems. A possible advantage of our proposed framework is that users can flexibly add in or modify their specific requirements at any time. To demonstrate the feasibility of our proposal, we built a prototype of an intelligent Personal Computer Configuration Advisor available on the Web to assist the general users in configuring their own PCs. Interestingly, our work opens up many new directions for future investigation including the improvement of our optimizer to handle more complicated users' requirements, and the possible uses of efficient learning algorithms such as the ID3 algorithm [2] to classify different user-defined configurations into useful examples to guide the search during optimization.

Keywords : Personal Computer Configuration Problems, Genetic Algorithms, Intelligent Web Applications.

I. INTRODUCTION

Similar to many well-developed Asian countries like Hong Kong or Singapore, many people, due to the influence of the popular "Do-It-Yourself" (DIY) philosophy from western society, are keen in configuring and building their own personal computers (PC) to suit their own requirements, often as the first step to learn to use software or access the Internet. Clearly, the PC configuration problems are widely occurring decision problems faced by many people in which people are always interested to know the optimal, or possibly sub-optimal, PC configuration within their limited budget. However, since the PC technology nowadays are changing very quickly, the diversity of PC hardware components, such as the different types of processors, random-access memory (RAM) and the motherboards, and their limited compatibility between certain components due to the underlying proprietary manufacturers often complicates the whole decision problem, thus making it difficult to handle by the general public.

In fact, many configuration problems [2] as in the area of computer-aided design or manufacturing (CAD/CAM) [6] are well studied. In particular, there were some interesting research work [6] on formulating the machine configuration problems formally as discrete optimization problems, and then applying local search methods such as genetic algorithms [11,12] or simulated annealing [9,10] to handle these configuration problems successfully. The previous research experience reported in [6] already revealed that formulating the machine, or any general, configuration problems as constrained optimization problems (COPs) will definitely result in systematic handling of users' requirements as constraints or optimization criteria. At the same time, the users can flexibly add in or modify their requirements at any time to see the resulting configurations for planning or control [5]. Thus, in this paper, we firstly gave a formal definition of these PC configuration problems as discrete optimization problems. More importantly, based on this formal definition, we proposed a systematic and flexible framework for solving these difficult real-world discrete optimization problems.

Basically, a discrete optimization problem involves a set Z of variables, each variable V_i ($\in Z$) with a finite domain D_i of discrete values, a set C of constraints on some subsets of variables limiting the combination of values assigned to those involved variables, and a set of objective functions f_j for minimization/maximization. The challenge is to find a globally optimal solution which minimizes/maximizes all the objective functions f_j while satisfying all the constraints in C. Mathematically, the discrete minimization problem can be specified as follow.

$$\min \sum_{j=1}^{m} f_j \qquad \textit{subject to} \quad \forall c \in C \quad cf(c) = 0 \ldots\ldots\ldots\ldots\ldots\ldots (1)$$

where m denotes the total number of objective functions in the problem, and *cf(c) is an arbitrary function* which returns 0 when the specific constraint c is satisfied. Otherwise, it returns 1. Clearly, to solve maximization problems, we can simply negate all objective function f_j in the above formulation as $\min \sum_{j=1}^{m} -f_j$. In handling these optimization problems, which are always *NP-complete* [3,4], one of the frequently used heuristics is the branch-and-bound (B&B) heuristic [1,2] in which the exploration of any partial solution in a search tree will be abandoned immediately whenever the search cost of that partial solution, as represented by an arbitrary objective function, already exceeds the minimal cost for the optimal solution found so far. For instance, the B&B heuristic has been successfully applied to handle the famous traveling salesman problems [2,3] to guarantee the finding of the shortest path to transverse all the required cities in one round trip. However, in the worst cases, the B&B search method may still require exponential time to find out the optimal solution for any general COP. Besides, in some cases [10] even with the aid of useful heuristics, finding a feasible solution satisfying all the constraints in *C* is still very difficult. Thus, the users may relax some constraints in *C*, and ready to accept some "partial solutions" [5] representing the optimal or sub-optimal to the objective functions. On the other hands, there are some real-life COPs which are sparsely constrained, thus having too many feasible solutions. In those cases, it will then become difficult to guarantee global optimality of the resulting solution within a reasonable period of time. Thus, the users may also be willing to accept a sub-optimal solution as a trade-off for efficiency. In general, many real-life PC configuration problems belong to this latter case of discrete COPs in which given the diversity of possible PC configurations as feasible solutions, the users will often accept a sub-optimal solution when the resulting configuration can be returned quickly from the optimizer.

Therefore, based on the discrete COP formulation, we proposed a flexible and systematic framework to handle the PC configuration problems in which most of the useful information about the PC hardware components is often stored in some database files locally in different computer companies. Usually, these companies will also make use of the information already stored in those database files to advertise the products on their company Web pages. Accordingly, inside our proposed framework, we consider the optimization of options for PC configurations based on information provided from a centralized database system. It should be noted that the assumption of a centralized database system is generally valid since nowadays, most of the database systems we used in companies are often compatible with the de facto Open DataBase Connectivity (ODBC) standard [7]. In general, the heterogeneous database systems linked up via the Internet and the ODBC interface layer can be regarded a centralized database system when we ignore the latency time mainly due to the network communication and the possibly inconsistent data stored in these different database systems. Nevertheless, based on the presumably consistent data about PC components stored in a centralized database system, there are two major approaches we used to efficiently optimize the ultimate PC configurations to satisfy the users' requirements for handling these specific instances of real-life COPs. First, similar to the branch-and-bound method [1,2], we prune off any alternative choice which already exceeds the planned budget during each search step. In addition, we include a constant threshold value *n* to monitor the size of the possible PC configurations we will consider in each search step so as to avoid the combinatorial explosion problem frequently occurring in solving these PC configuration problems given the diversity of some particular PC components. Clearly, since we are not performing an exhaustive search, the approaches we used can be efficient but may not be able to guarantee the global optimality of the resulting configurations.

To demonstrate the feasibility of our proposed systematic framework, we firstly collected the actual data from some major computer shopping centers in Singapore to build our own centralized and local databases of PC components for investigation, and then used the widely available Practical Extraction and Report Language (PERL) [7, 8] Version 5.0 to implement the afore-mentioned preliminary search strategies to handle these practical PC configuration problems. The empirical experience of using our proposed approach to handle the PC configuration problems is fairly encouraging. Furthermore, the resulting optimizer is used to build a platform-independent prototype of the useful Web-based Personal Computer Configuration Advisor so as to facilitate the general users to quickly set up their required PC configurations. Obviously, our work opens up many new directions for future investigation such as improving our current optimizer to handle more complicated users' requirements, and the possible uses of efficient learning algorithms such as the ID3 algorithm [1,2] to classify different user-defined configurations into useful examples to guide the search during optimization.

This paper is organised as follows. Section 2 describes the PC configuration problems as specific instances of discrete COPs, thus forming a systematic framework for optimizing choices of PC components to satisfy the users' requirement. In Section 3, we detail and justify our proposed search strategies to handle these specific COPs according to their unique problem structures. Section 4 provides the empirical evaluation of our cross-platform prototype implementation of the proposed optimizer in terms of efficiency and costs of the resulting configurations, with an example application of our optimizer to build a Web-based PC Configuration Advisor. Lastly, we conclude our work in Section 5.

2. PC Configuration Problems

Basically, the PC configuration problem is to select a configuration of personal computer hardware parts to build a complete system, taking into account the *compatibility* issues between the different hardware components. For instance, Intel Pentium II CPU should be attached to a Slot-1 Motherboard. Definitely, one of the most important optimization criteria in many real-life situation is *price*. Thus, a general formulation of the PC configuration problems as discrete COPs can be as follows.

$$\min \sum_{j=1}^{m} \mathrm{cost}(Pj) \qquad \textit{subject to} \qquad \forall comp(Pi,Pj) \in C \quad cf(comp(Pi,Pj)) = 0 \,..........(2)$$

In *(2)*, cost(*Pj*) denotes the cost for the component *Pj*, *C* specifies all the compatible relations (constraints) between the components *Pi* and *Pj*, and the arbitrary function *cf* returns *0* when *comp(Pi, Pj)* is evaluated as true. For example, when the variables for the CPU and motherboards are : P_{CPU} = *"Intel PII CPU"* and P_{MB} = *"Slot-1 Motherboard"*respectively, *comp*(P_{CPU}, P_{MB}) = *true*, then *cf*(*comp*(P_{CPU}, P_{MB})) = *0*. Clearly, given the above general COP formulation, it is flexible to add in or modify the users' requirements specified as constraints or optimization criteria. For instance, when new components P_X and P_Y are added into the PC market, it is easy to simply add a new constraint *comp*(P_X , P_Y) into *C* to store their compatibility information.[1] Besides, for more complicated real-life cases, we can simply extend the minimization function to consider other important factors such as a weighting value for each component to reflect the users' preference.

Variables

The variables *Pi's* of interest for the PC configuration problem is the set of hardware components which will be assembled to build a working personal computer system. Below is the table of the variable names which we used to formulate a typical PC configuration problem and their short descriptions for clarity.

Variables	*Short Descriptions*
CPU	Central Processing Unit. The core of a computer. For simplicity, only uni-processor computers are considered in this problem. Needs to be compatible with Memory and Mainboard.
Memory	The hardware where data is temporary stored. Need to be compatible with CPU and MainBoard.
MainBoard	The PVC-type board which all the various hardware components are connected together. Most of the constraints are defined on this variable.
Casing	The covering of the computer. No constraint.
DisplayCard	The hardware component which handles the processing of the graphical display. No constraint.
HardDisk	Mass storage device. No constraint.
CD_ROM	Read-only input media. DVD and CD Writer/Rewriter do not belong to this variable.
Modem	The interface to telephone whereby the computer can be connected to other computers over the PBX-type modem. No constraint.
Monitor	The display unit. No constraint.
Printer	The printing device. No constraint.
Scanner	The scanning input device. No constraint.
SoundCard	The device which handles the processing of audio input/output. No constraint.

Table 1. Typical Variables involved in PC Configuration Problems

Clearly in the above table, there are about 12 common variables usually involved in configuring a PC nowadays. The first three variables, namely CPU, Memory and MainBoard, of the table represent the most important variables which are often highly constrained. The remaining nine variables are totally un-constrained. Thus, this special relationship between the PC components depicts the unique problem structure of the PC configuration problem : the constraints between the first 3 variables will leave very few combinations of options to be matched against a possibly large number of possible choices in a later stage of the search. It should be noted that this large possibility occurred in the later search stage may pose major difficulty to most optimizers such the branch-and-bound techniques since no other available information can be used to prune off any value during the search.

[1] Clearly, some readers may argue that these compatibility constraints are in fact no different from the logical relations like "father(john, mary)" to specify "john" is the father of "mary" in some logic programs. Of course, we would agree on that since it is generally true for most constraints. However, the "encapsulation" of these general relations between objects as a specific constraint will in general provide *a more systematic way* to handle that particular kind of constraints. As a result, we will discuss how a special-purpose search algorithm (as constraint solver) can be designed to handle those compatibility constraints *more efficiently* for our PC configuration problems in the next section.

Domain

Table 2 shows the actual domain size and the range of integer values stored for each variable.

Variables	Size	Range	Variables	Size	Range
CPU	40	[0..39]	**CD_ROM**	26	[0..25]
Memory	46	[0..45]	**Modem**	32	[0..31]
MainBoard	69	[0..68]	**Monitor**	105	[0..104]
Casing	14	[0..13]	**Printer**	34	[0..33]
DisplayCard	110	[0..109]	**Scanner**	42	[0..41]
HardDisk	82	[0..81]	**SoundCard**	25	[0..24]

Table 2. The Actual Domains for Variables in PC Configuration Problems

The average domain size for all the variables is 52. Thus, the size of the search space for all possible combinations of PC components is roughly of the order of 52^{12}. More importantly, this huge search space has lots of possible sub-trees which cannot be further pruned off by any heuristic. Accordingly, the conventional enumerative search [5] may simply give unsatisfactory performance to return the globally optimal PC configuration. Therefore, in the next section, we will discuss our proposed search strategy which sacrifices global optimality for efficiency to tackle these real-life COPs.

Furthermore, it should be noted that these PC, or possibly other, configuration problems are very much different from some famous routing problems such as the traveling salesman problem, on which the branch-and-bound (B&B) strategy performs reasonably well, in terms of the underlying domains for each variable. For a traveling salesman problem with *12* cities, the underlying domains for all the variables $V_1..V_{12}$, denoting the first to the last city to be visited on a trip, are the same. That is the range [1..12] to represent the **same** *12* cities. Thus, at each search step, we can use the B&B heuristic to prune off any alternative route starting and ending with the same cities, say "city 1 - ......- city 9", but with a higher total distance covered. However, in our PC configuration problems, the possible choices at each search step will be totally different from the previous choices already occurred in the previous search steps. Thus, the B&B heuristic may not necessarily be a useful heuristic in solving the PC configuration problems.

Constraints

The following list shows some example of the constraints frequently occurred in the PC configuration problems. The first one specifies that the total cost of the configuration must be less than or equal to the budget predefined by the user. The second and third constraints are obviously about the *CPU, Memory* and *MainBoard.* The second one states that the type of the *CPU* must match with the type of the *MainBoard* while the last one specifies that the speed of the *MainBoard* and the *Memory* must be the same.

- TotalPrice <= Budget
- CPU.Type = Mainboard.Type
- Mainboard.Bus_Speed = Memory.Bus_Speed

Clearly, it is straightforward to add in more constraints as users' new requirements. As in most PC configurations, the cost and performance are two common factors for consideration.

3. OUR PROPOSED FRAMEWORK & SEARCH STRATEGIES

In this section, we will firstly look into a systematic framework for integrating different optimizers to handle the PC configuration problems based on the formulation of discrete optimization problems which we discussed in the previous section. Figure 1 shows our proposed systematic framework for integration of different optimizers.

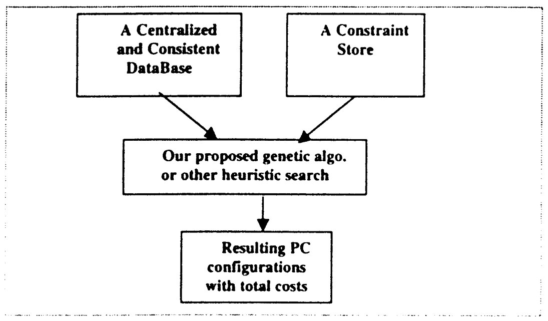
A Centralized and Consistent DataBase
A Constraint Store
Our proposed genetic algo. or other heuristic search
Resulting PC configurations with total costs

Fig. 1. Our Proposed Systematic Framework for Optimizing PC Configurations

It should be noted that in our systematic framework for constrained optimization to handle the PC configuration problems, we assume our search and optimization process have to performed on a set of consistent data from a centralized database system. In addition, there is a constraint store to provide useful information about the users' requirements as a collection of constraints. Based on these useful information, our optimizer with customizable search strategies will return a set of (sub-)optimal PC configurations with their total costs to the users according to the predefined optimization criteria. Clearly, other possible optimizers can also be added into our framework as some intelligent plug-in components for better efficiency or handling more complicated optimization criteria.

In the following, we will discuss our proposed optimizer with customizable search strategies which may not guarantee to always return the PC configuration with the globally minimal cost. However, as opposed to enumerative search, our proposal will definitely be more efficient. Table 3 shows the two main factors we used to control the search, together short descriptions to explain their roles. Both control parameters are predefined by the user of the search strategies.

Control	*Short Descriptions*
Budget	The budget for the whole personal computer system. Total prices of the components are not allowed to exceed the budget.
Threshold	Number of the best partial solutions to be considered in each search step

Table 3. **Our Control Factors to Search for (Sub-)Optimal PC Configuration**

Similar to the B&B heuristic, the predefined control parameter *Budget* will be used to filter out any partial configuration which already exceeds its allowed value. In addition, the *Threshold* value *n* will help to ensure the search will always return the best *n* partial configurations after sorting all possible partial configurations against their total prices at each search step. Based on these two strategies, we designed a beam-search based optimizer to solve these PC configuration problems as follows.

```
1. Retrieve and sort the values for the current variable by price.

2. Find the next matching value of the variable in the sorted value queue.
Check that all the constraints are satisfied for this value. This includes the
budget constraint (that is Total_price <= Budget).

3. If no constraint is violated, goto 6.

4. Else if queue is not empty, repeat 2. Else Fail.

5. Set the current solution set to include this value.

6. If number of partial solutions < threshold, repeat 2. Else 7

7. If variable queue is not empty, proceed to set current variable to next
variable in the queue. Repeat 1.

8. If number of solutions > 0 then report Solved. Else report Failure.
```

Our second proposal is a min-conflict heuristic (MCH) based micro-genetic algorithm (MGA) [6], that is an evolutionary algorithm with small population size of chromosomes, to handle the PC configuration problems. In general, when handling constraint satisfaction or optimization problems with evolutionary algorithms, a bit-string called chromosome will usually be used to represent a possible valuation for all the variables in the constrained problem. When the valuation represented by a particular chromosome violates no constraints in the problem, the chromosome denotes a solution. For solving constrained problems with a large number of variables (say > *500*), a MGA with a small population size *PZ* in the range of *6-20* may often outperform the traditional evolutionary algorithms with larger population size [6]. This is probably because the computational cost can be minimized by focusing the search only on a reasonably small population of chromosomes without much impact on the search efficiency.

The min-conflict heuristic is to consider modifying only a single variable at a time, and to assign a value to that variable which is locally minimum in terms of constraint violations. When there are several local minima (ties) in terms of constraint violations, a value will randomly be selected among these ties. In our previous work [12], we proposed a mutation-based evolutionary search scheme (*MCH-MGA*) to efficiently solve constraint satisfaction problems (CSPs) [3] as follow.

```
MCH-MGA(CSP, fitness(), PZ, MAX_GENS) {

  initialize the 1st generation of PZ chromosomes randomly;

  set best_fit = best fitness value of the current population;
  set ngens = 1;

  while (best_fit != 0 && ngens < MAX_GENS) {
    uses selection scheme to choose the pivot_gene;
    applies MCH-mutate to pivot_gene;
    applies descent-mutate to other genes;
    if (population is in local minimum) then applies popu-learn;
    update best_fit;
    ngens++;
  }
  if(best_fit == 0) return solution;
  return failure;
}
```

Clearly, the *MCH-MGA* depends on the selection scheme, either *update-select* or *usage-select*, to pick up the most important variable/gene for the current search step to apply *MCH-mutate* in order to get the greatest improvement if possible. For the remaining variables, it generally applies the more efficient operator *descent-mutate*. The *MCH-mutate* is basically a MCH variable updating operator while the *descent-mutate* operator is a restrictive operator which allows only mutations resulting in decreases in the total number of constraint violations. The *popu-learn* operator is simply an adaptation of the heuristic learning mechanism discussed in [6] to a population of chromosomes rather than a single chromosome. For detail, refer to [6].

To handle the PC configuration problems, we have to firstly modify the original *fitness* function to combine the cost of constraint violations with actual *cost* of the different PC hardware components P_j*'s*. Assume the average cost for each PC component P_j is about *1000*, we can adjust the *fitness* function to an arbitrary and relatively larger weight such as *10, 000* associated with the part for constraint violations as follows.

$$\min \sum_{j=1}^{m} (\mathrm{cost}(Pj) + 10{,}000 * \sum_{\forall comp(Pi,Pj) \in C} cf(comp(Pi,Pj)))\ldots\ldots\ldots\ldots(3)$$

Thus, the ultimate effects produced by *(3)* on the evolutionary search will help to avoid any possible constraint violation. It should be noted that with this new formulation, it is straightforward to add in the users' preferences for certain PC components by adding another weighted term, for example *1000 * user_pref(Pj)*, into the above *fitness* function. Besides carefully adjusting the *fitness* function, we have to modify the testing condition for the *while-loop* in the *MCH-MGA* to : *while (ngens < MAX_GENS)* since we are now handling optimization problems instead of CSPs. Therefore, the unsatisfiability testing condition: *best_fit != 0* should be removed. Furthermore, the last two statements of the *MCH-MGA* procedure should also be modified to always: return (sub-)optimal solution for the optimization problems at hands. After putting into all these minor modifications to the above *MCH-MGA* procedure, we rename this new MCH-based evolutionary algorithm for solving optimization problems as *MCH-MGA-OPT*. In the following section, we are going to compare the performance of *MCH-MGA-OPT* against the beam-search based optimizer in solving the real-life PC configuration problems.

4. EMPIRICAL EVALUATION

Based on the above algorithms for our proposed optimizers, we built a prototype in PERL Version 5.0 since we planed to integrate our optimizer to build a Web-based PC Configuration Advisor to facilitate the general users to configure their own PC. To evaluate the performance of our prototype for the proposed optimizer to handle the PC configuration problems, we focused on two main aspects of the computational results returned by our optimizer in the following analysis. The first important factor to consider is the quality of the solution in term of the total costs of the PC configurations for minimization. The second factor of interest is the efficiency of our optimizer as reflected by the timings in CPU seconds. Accordingly, we run our prototype 10 times for different settings of threshold values on a DEC-Alpha workstation running Digital Unix Version 4.0.8.

The table below shows a sorted listing of the total costs of the different PC configurations, labeled from 01 to 30, returned by our beam-search optimizer for *Budget* <= $*3000* and *Threshold = 30*.

Cfg Cost	Cfg Cost	Cfg Cost	Cfg Cost	Cfg Cost
01 753	07 757	13 763	19 765	25 766
02 755	08 759	14 763	20 765	26 767
03 755	09 760	15 763	21 765	27 767
04 755	10 762	16 764	22 765	28 768
05 757	11 762	17 764	23 765	29 769
06 757	12 763	18 764	24 766	30 769

Clearly, there is a fair difference of SG $16 between the first minimal total cost and the last total cost of the configuration 30 as shown in the above table. In addition, it should be noted that from our initial experiments, the top 10 configuration-cost pairs usually remain unchanged for the same budget limit even though we changed the threshold value from 30 to 20 and then to 10. This demonstrated the stable performance of our optimizer in handling these PC configuration problems.

Table 4 shows the performance of our beam-search optimizer for handling the PC configuration problems with *Budget* <= $*3000* and varying the *Threshold* value at *1, 5, 10, 20* and *30*. For each case, the reported data is the average CPU time in seconds over *10* runs.

Threshold	*Average CPU time in seconds (10 runs)*
1	*3.18*
5	*3.19*
10	*3.32*
20	*3.40*
30	*3.51*

Table 4. Performance of our Optimizer for Different Threshold Values

The above table clearly showed that our optimizer is fairly efficient in handling a typical PC configuration problem with different threshold values to control the search. Also, the performance of our optimizer is compact and stable since there is only slight increase in the average CPU time from 3.18 to 3.51 seconds when the corresponding threshold value is changed from *1* to *30*.

MAX_GEN	**20**	**50**	**100**
PZ = 1	0.35s (847.2)	0.71s(715)	1.28s(715)
5	1.26s(716.8)	3.10s(715)	6.04s(715)
10	2.38s(715)	5.99s(715)	12.03s(715)
20	4.76s(715)	11.97s(715)	24.07s(715)

The above table gives the performance of the *MCH-MGA-OPT* for *PZ = 1, 5, 10 and 20* and *MAX_GENS = 20, 50 and 100* for comparison. For each table entry, the first data represents the average CPU time in seconds while the second one in parentheses denotes the cost in SG$. When we compare against the quality of solutions as returned by the beam-search based optimizer, it is clear that the *MCH-MGA-OPT* consistently outperformed the beam-search based optimizer with the optimal cost = SG $715 for most cases. In fact, as for *MCH-MGA-OPT*, the minimal cost over *10 runs* consistently stays at SG $715 for all the cases, which probably represents the global optimal solution[2]. In general, the beam-search based optimizer could only better the average cost returned by *MCH-MGA-OPT* in the weakest case where *PZ=1 and MAX_GEN = 20*. On the other hand, the *MCH-MGA-OPT* generally outperformed the beam-search based optimizer in terms of solution quality.

Since the performance of our proposed optimizers from the initial experiments is satisfactory, we integrated our optimizer as a plug-in component into our targeted Web-based PC Configuration Advisor for optimizing different PC configurations according to users' budget. Firstly, our Web-based PC Configuration Advisor supports rule-based information processing to actively anticipate or validate users' inputs, and also analyze hardware information stored in databases to provide useful advice on the best PC configurations for certain usage subject to a user-defined budget limit, and a pre-defined threshold limit used during the search. Specially, we allows the active uses of rules, as stated in the rule script to represent some domain experts' knowledge, to "dynamically" prune off options within the same page (intra-page) or between the pages (inter-page). For example, when the user selects the "Type of PC" to be configured is "Server" on the first page, the option of "17-inch monitor" will be automatically removed in the last page[3] with the active use of the domain experts' rules. Besides ensuring valid inputs from the users, it is mainly used to remove irrelevant choices so as to facilitate the search efficiency. After all possible pruning, our proposed optimizer, implemented as a easy-to-modify plug-in component inside our PC Configuration Advisor, then performs the controlled (*best-n)* search according to the predefined budget limit and threshold values during optimization of PC configurations. Thus, for a user's requirement like *budget* $\leq$ *3000 and threshold = 20*, our Web-based PC Configuration Advisor will definitely return only the first *20* optimal PC configurations with the total cost less than $*3000*. In case where no PC configuration can meet the specified budget, an error message will be printed on the resulting Web page.

[2] Of course, this will require a complete solver to later verify the global optimality of our solution found.

[3] Technically, this special feature is achieved with JavaScript through the uses of cookies to remember some important values for each page.

5. CONCLUSION

It is undeniable that the PC configuration problems are both practical and interesting optimization problems widely occurring in many well-developed countries of the world. However, specific configuration problems are interesting since they have the unique problem structures that some components are highly constrained while the rest are totally unconstrained. This may pose challenge to some conventional enumerative search methods. As the PC technology is likely to be more complicated in the future, it will definitely become more difficult to handle these specific configuration problems. In this paper, we gave a formal definition of the PC configuration problems as discrete optimization problems. Based on this problem formulation, we proposed a systematic framework which allows the integration of different optimizers for optimizing the PC configurations according to the useful information stored in a centralized database and constraint store. More importantly, after analyzing the unique features of the PC configuration problems which may possibly lead to combinatorial explosion during the search, we proposed two useful search strategies, one evolutionary algorithm and one beam-search, to control the search and optimization process in our optimizer. The min-conflict heuristic based MGA optimizer compared favorably against the beam-search method in handling these real-life PC configuration problems. To demonstrate the feasibility of our proposals, we implemented prototypes of our proposed optimizers for some empirical evaluation, and later integrated it into a Web-based PC Configuration Advisor to facilitate the general public in configuring various PCs.

Clearly, there are several interesting directions for our future investigation. One obvious direction is to try our optimizer on many different cases, and also include other optimizers such as the B&B method for a complete comparison. Second, it would be interesting to improve our optimizer to handle more complicated constraints and optimization criteria to handle more complex real-life PC, or other general, configuration problems. Lastly, we are currently studying the possible uses of efficient learning algorithms such as the ID3 algorithm [2] to classify different user-defined configurations into useful examples so as to guide the search during the optimization process.

Acknowledgement

We are grateful to Mr. W.K. Foo for his valuable help in implementing the beam-search based optimizer. In addition, we would like to acknowledge the ARF grant (ref. no. RP3991612), the National University of Singapore.

6. REFERENCES

[1] "Artificial Intelligence : A Knowledge-Based Approach" by Morris W. Firebaugh, PWS-Kent Publishing Company, Boston, 1988.
[2] "Artificial Intelligence" by Elaine Rich and Kevin Knight, McGraw-Hill International Edition, 1991.
[3] "Discrete Mathematics – A Unified Approach" by Stephen A. Wiitala, McGraw-Hill International Edition, 1987.
[4] "Introduction to Algorithm" by Thomas H. Cormen, Charles E. Leiserson and Ronald L. Rivest, The MIT Press, 1994.
[5] "Foundations of Constraint Satisfaction" by Edward Tsang, Academic Press, 1993.
[6] "Genetic algorithms versus simulated annealing : Satisfaction of large sets of algebraic mechanical design constraints" by A.C. Thornton, in *Proceedings of Artificial Intelligence in Design*, pp. 381-398, 1994.
[7] "Discover PERL 5" by Naba Barkakati, IDG Books WorldWide Inc., 1997.
[8] "Programming in PERL" by Larry Wall, O'Reilly, 1995.
[9] "Boltzmann machines for traveling salesman problems" by E. Aarts and J. Korst, *European Journal of Operational Research*, 39:79-95, 1989.
[10] "Optimization by simulated annealing : an experimental evaluation; Part II, graph coloring and number partitioning" by D. Johnson, C. Aragon, L. McGeoch, and C. Schevon. Operations Research, 39(3)378-406, 1991.
[11] "Solving small and large scale constraint satisfaction problems using a heuristic-based microgenetic algorithm" by G.Dozier, J. Bowen and D. Bahler. *In Proceedings of the IEEE International Conference on Evolutionary Computation*, 1994.
[12] "Improving Evolutionary Algorithms for Efficient Constraint Satisfaction" by Vincent Tam and Peter Stuckey, *International Journal on Artificial Intelligence Tools*, Vol. 8, No. 2, World Scientific Publishers, December 1999.