Systemic behavior of cooperative search algorithms

Michel Toulouse

Department of Computer Science,
University of Manitoba
Email: toulouse@cs.umanitoba.ca

Teodor Gabriel Crainic

Departement management et technologie
Université du Québec à Montréal
and
Centre de recherche sur les transports
Université de Montréal
Email: theo@crt.umontreal.ca

Brunilde Sansó

École Polytechnique
and
Centre de recherche sur les transports
Université de Montréal
Email:bruni@crt.umontreal.ca

Keywords: Cooperative search algorithms, distributed computing, parallel meta-heuristics, emergent computing.

Abstract

Distributed computer systems may be used in various ways to find good solutions to difficult combinatorial optimization problems. An interesting approach consists in executing concurrently different search methods that exchange information gathered in previously explored regions of the solution space. This cooperation mechanism strongly impacts how the solution space is explored. We introduce a formal representation of such cooperative procedures based on discrete-time dynamical systems. We describe how the search behavior of cooperative programs depends on systems of complex and correlated interactions. We derive the notion of systemic cooperation and illustrate through simulations the relevance of this notion to the understanding of the search behavior of cooperative procedures.

1 Introduction

Several search methods, based on exhaustive, local or random exploration of the solution space, use data collected from previously explored regions of the solution space (the search history) to complement their basic approach. This is the case, for example, with Branch-and-Bound, which keeps track of the best bounds to reduce the magnitude of the exhaustive search; Tabu Search (TS), which keeps track of attributes of previously visited solutions as a mean to guide the local search; Genetic Algorithms (GA), which use the fitness of individuals previously met in the population to bias the random selection of the parents; etc. Search history is also used to design parallel (distributed) algorithms for these as well as other methods. For a particularly important category of such parallel algorithms, data gathered by a sequential search program is recycled and shared with other concurrently executing search programs. In the literature, these parallel algorithms are referred to as *cooperative search* [3,4,19,20], *multiple interacting walks* [40] or *population approaches* [28]. Procedures such as coarse-grained parallel genetic algorithms (PGA) [17,33,35] and fine-grained PGA [16,26,30], Memetic Algorithms [6,27,29], parallel Branch-and-Bound [13], tabu search algorithms [8,10,11,23], simulating annealing procedures [22,24,25], and special purpose Artificial Intelligence (AI) cooperative searches [4,19] are examples of parallel search methods based on cooperation born from sharing gathered information among several sequential search programs.

The primary rationale behind sharing information among search programs is to speed up the exploration of the solution space. However, several researchers have observed that cooperation by sharing information"was more than just hardware acceleration" (Cohoon & all [5]). Researchers in the field of parallel genetic algorithms (Tanase [35,36], Jog and Gucht [21], Cohoon & all [5], Munetomo, Takai & Sato [31]) concluded that cooperation not only produces speed-up, but it also profoundly modifies the search pattern of the cooperating programs. They found empirical evidence that the specification of how search processes cooperate has an impact on how PGA succeed in finding good solutions. Similar results have been obtained in many other fields [7]. However, while researchers acknowledge that cooperation has an impact on the search pattern of cooperating search processes, only limited efforts to develop a comprehensive model of cooperation among search processes (Huberman [20], Clearwater & all [3]) have been undertaken so far. This is reflected, for example, in the way cooperative algorithms are designed, often simply by adding communication extensions to existing sequential search programs. We believe these ad hoc approaches fail to identify important design opportunities in cooperative algorithms. To contrast, consider that even though algorithms for cellular automata and neural networks are specified in terms of cooperation mechanisms among trivial computational elements, their design and analy-

sis takes into consideration macroscopic behaviors such as self-organization, phase transition and stabilization toward attractors. In this paper, we show that similar considerations can be used for designing more efficient cooperative search algorithms.

In the context of cooperating search programs, we identify as macroscopic behavior any form of behavior occurring because of interactions among two or more search programs. For example, the best solution found by two cooperating search programs is a macroscopic behavior of the cooperative procedure if this best solution is not the same as when both the same programs are run without cooperation. The sharing of information among search programs gives rise to several forms of macroscopic behaviors. In this paper we focus on *systemic cooperations*, a macroscopic behavior involving correlated interactions, i.e., interactions that are triggered by the occurrence of other interactions. Systemic cooperations are dynamic phenomena that encompass several cooperating search programs. They create control dependencies among cooperating search programs which have a significant impact on the search pattern of cooperating programs. We believe that a better understanding of this type of macroscopic phenomenon will help to improve the design of cooperative algorithms.

In order to make a precise description of these phenomena, we have modeled cooperative procedures as discrete-time dynamical systems. The global state description of these systems contains all relevant information showing how systemic cooperations build-up and how they impact the search pattern of cooperating programs. We also carry out simulations that describe different cooperative procedures with the same set of sequential programs, initial conditions, and set of search parameter values, which nevertheless exhibit completely different search patterns under the influence of systemic cooperations.

The organization of the paper is as follows. In Section 2, the tabu search meta-heuristic used in the formal argument and the experimentations is described. In Section 3, some aspects of cooperative procedures are briefly introduced. Our framework of cooperative algorithms is presented in Section 4. In Section 5, we introduce the notion of systemic cooperation. The tests are described in Section 6. Conclusions are presented in Section 7.

2 Tabu Search

Tabu Search is an iterative meta-heuristic aimed to search for good solutions to combinatorial optimization problems. A tabu search algorithm is actually an extra layer above a local search procedure; therefore, we first introduce the concept of local search in a solution space.

A local search algorithm presupposes the definition of a *neighborhood* structure and the corresponding *moves* in the solution space of the optimization problem. A move is a modification that transforms a solution $x \in X$ (where X is a set of feasible solutions) into another solution y (the pivot of the simplex method is an example of a move). The solutions that can be obtained by a single application of a move to a given solution x are considered to form the neighborhood $N(x)$ of this solution. A local search process starts with an *initial solution* and looks in its neighborhood for an improving solution. If one is found, it replaces the current solution and the search resumes with the new current solution. The execution of a local search program can be represented by an oriented graph, where nodes represent solutions and arcs stand for moves. The sequence of solutions visited during the execution of the program forms an oriented path, called the *search path* (see Fig. 1).

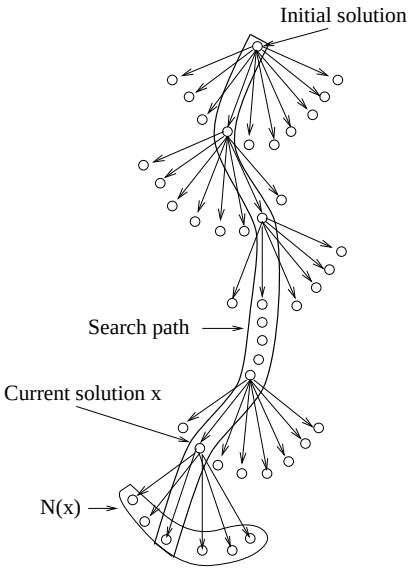


Fig. 1. Oriented Graph of Local Search Trajectory

Local search algorithms generally accept either the first or the best improving solution in the neighborhood of the current solution. When no improving solution can be found in the neighborhood of the current solution, it is a local optimum according to the neighborhood structure in use. More complex search methods often use far more elaborate schemes to choose the next current solution. This is one of the characteristics of tabu search. If trapped in a local optimum, the tabu search method allows non-improving moves in order to move away from it and continue the exploration of the solution space.

Tabu search keeps records of the attributes of the solutions visited in the search path via different *tabu memories*. The contents of the tabu memories is then used as input to the process of choosing the next current solution at each iteration or to select a new neighborhood. Thus, some of the most recently visited solutions in the search path are excluded from the neighborhood of

the current solution in order to prevent the search path from cycling. In other cases, solutions that are not in the current neighborhood are evaluated as part of a long-term *search strategy*, since tabu search uses different neighbourhoods of various complexity to support different phases of the search: local search, intensification, diversification, etc. For a more comprehensive description of the tabu search method, see [14,15].

Schematically, a tabu search program executes the steps displayed in fig. 2 each time it selects the next solution.

(1) Evaluate the solutions in the neighborhood $N(x)$.
(2) Select a solution $x' \in N(x)$ given according to the rules of the current phase of the search.
(3) Make the current solution of the next iteration equal to x'.
(4) Update the different memories (and search parameters, eventually) of the tabu program.

The tabu memories are repositories for information on previously visited regions of the solution space. As such, their content has a direct impact on step 2 and the evolution of the search, both in a sequential and parallel cooperative context.

3 Cooperative search algorithms

Sequential search methods like simulated annealing, tabu search, branch-&-bound, and others are not simple to parallelize. The standard functional and data decomposition techniques can not obtain a large degree of parallelism due to the sequential dependencies between the iterative steps of these methods (see Greening [18] and Yannakakis [41] for this topic).

Cooperative search algorithms are not based on an explicit decomposition of the problem domain. Rather, parallelism is achieved by launching concurrently several independent search programs [1,11,22,34,40]. Different initial solutions and different values for the search parameters may be used for each independent search, which yields an *implicit* decomposition of the solution space. In this way, the solution space can be explored in parallel without being limited by the sequential dependencies of the search methods. Parallel search methods defined as above are sometimes identified as *parallel independent search* in the literature on parallel search heuristics. If p independent searches are executed concurrently, the parallel computation achieves a linear speed-up when each independent search executes $\frac{l}{p}$ iterations and the quality of the best solution found by the search programs is similar to the solution obtained by a single program run for l iterations.

The next step in the design of cooperative search algorithms consists in defining how the individual searches communicate and collaborate together. In this paper, we focus on procedures where cooperation is based on the sharing of gathered information. This cooperation is generally accomplished by setting values for at least four parameters: information gathered and made available for sharing; identification of programs that share information; the frequency of information sharing among the sequential programs; and the number of concurrent programs (which is also a parameter in the design of parallel independent search algorithms). We identify these four parameters as being part of the *system control parameters* since their values impact the organization of the cooperative search as a global entity, as opposed to the search parameters of each individual program which are primarily affecting the search pattern of the sequential programs as such. The existence of these parameters as part of the design of cooperation among sequential search programs has been discussed by Laursen [22], Mahfoud and Goldberg [25], and Toulouse, Crainic and Gendreau [37].

Cooperative search algorithms obtain speed-up similar to parallel independent search algorithms. The most noticeable impact of cooperation consists in the modification of the search paths of the cooperating search programs. One can measure the extend of these changes by comparing the same set of search programs embedded in an independent and a cooperative search procedure, respectively. Through these changes, cooperation could produce some extra speed-up if it can find in less than $\frac{l}{p}$ iterations a solution equivalent to the best solution found by the independent search procedure. Unfortunately, cooperative procedures are not always stable, they improve speed-up on some problem instances while degrading it on others. On the other hand, they often yield solutions of higher quality compared to independent search procedures. These observations point to the need to better understand how cooperation impacts search programs and control cooperation. This paper is a step toward this goal.

4 Formalization of cooperative search

In this section, we model a system composed of cooperating tabu search programs as a discrete-time dynamical system. We first define the concepts of iterative map and discrete-time dynamical system. We then represent the sequential tabu search, the parallel independent tabu search, and the cooperative search as iterative maps. The second representation borrows from chaotic relaxation algorithms [2,12] to account for the asynchronous behavior of independent programs. For the third, we enhance the representation of the iterative map to account for the interactions among the search programs. This leads to the representation of cooperative procedures as discrete-time dynamical

systems.

4.1 Iterative maps and discrete-time dynamical systems

Let ψ be a p-dimensional state vector, and f some iterative operator. An iterative map has the structure:

$$\psi(k+1) = f(\psi(k)), \quad k = 0, 1, 2, \ldots \tag{1}$$

where the state vector $\psi(k+1)$ depends only on the state vector $\psi(k)$ and the operator f (f is considered to be invariant over k). If ψ and f are separable in p components, one can rewrite (1) as

$$\psi_i(k+1) = f_i(\psi_1(k), \psi_2(k), \ldots, \psi_p(k)) \tag{2}$$

where $\psi_i(k)$ is the i-th component of $\psi(k)$ and f_i is the i-th component of the operator f.

If no interaction takes place among the p components, equation (2) becomes $\psi_i(k+1) = f_i(\psi_i(k))$ for $i = 1, 2, \ldots, p$, and these equations do not define a dynamical system. Otherwise, equation (2) defines a discrete-time dynamical system.

4.2 An iterative map for the tabu search method

In this section we represent the aspects of the sequential tabu search method that are relevant to model the global dynamics of a cooperative tabu search procedure.

The sequential tabu search is represented as a non-separable iterative mapping such as (1). The state ψ has two constituents: the current solution $x(k)$ and the information gathered on the solution space $h(k)$, where k is an index representing the iterations of a sequential TS program and $h(k)$ is a data structure representing the content of the different tabu memories at iteration k.

At each iteration of the tabu search method, the operator f must update the two constituents of the state vector. Consequently, operator f is a tuple (s, v) of functions. Function s represents the search strategy of the tabu program and takes $h(k)$ and $N(x(k))$ as arguments, where $N(x(k))$ is the neighborhood of the current solution $x(k)$. This function returns the solution $x(k+1)$ of

iteration $k+1$. Function υ takes the values $x(k+1)$ and $h(k)$ as arguments and returns $h(k+1)$. Function υ updates the data structure $h(k)$ according to the attributes of the new solution $x(k+1)$ and the state $h(k)$ of the tabu memories at iteration k. At each iteration, a subset $\mu \subseteq \{1, 2, \ldots, m\}$ of the m fields of $h(k)$ is updated using information from the attributes of solution $x(k+1)$.

4.3 Iterative map of the parallel independent search programs

The main concern in developing a model to represent the global state of a set of parallel independent tabu search programs is the handling of the asynchronous evolution of these individual programs.

Let $CP = \{P_1, P_2, \ldots, P_p\}$ be a set of p sequential tabu search programs and $s_1, s_2, \ldots, s_p$ the set of corresponding search strategies, where $s_i \neq s_j$, if $i \neq j$ (i.e., each TS program applies a different search strategy). Let x_i be the current solution of program P_i and h_i the current state of the tabu memories of program P_i. In an independent search procedure CP, the P_i programs do not share gathered information. Therefore, for each sequential program P_i, the new current solution $x_i(k+1)$ obtained from the neighborhood $N(x_i)$ of solution x_i at iteration k (step 2 of a tabu iteration) is given by:

$$x_i(k+1) = s_i(N(x_i(k)), h_i(k)) \tag{3}$$

that is, the choice of the current solution at iteration $k+1$ depends only on the state of program P_i. The update of the tabu memories is given by:

$$h_i(k+1) = \upsilon_i(x_i(k+1), h_i(k)). \tag{4}$$

The value of $h_i(k+1)$ corresponds to the new state of the tabu memories of program P_i at iteration $k+1$, which results from updating the fields of $h_i(k)$ at iteration k using attributes of solution $x_i(k+1)$. Together, equations (3) and (4) are equivalent to the system (2), since x_i and h_i are constituents of ψ_i, while s_i and υ_i are sub-functions of f_i. By definition, no information is shared among parallel independent tabu search programs, consequently, equations (3) and (4) can be combined to form:

$$\psi_i(k+1) = f_i(\psi_i(k)), \;\; k = 0, 1, 2, ... \tag{5}$$

where f_i is applied to the single component ψ_i of the p-dimensional vector ψ.

The time required to perform one iteration of the tabu search method can vary considerably from one search program to another; sequential programs can run

on computers with different speeds or workloads. Consequently, one cannot index the global state of a set of tabu search programs using the value k of the iterations of each individual program. We introduce a few transformations to equation (5) to iterate on a discrete time scale rather than on the values of k.

Let $x_i(0)$ be the initial solution of program P_i. Let $\mathcal{S} = (\{\gamma(t)\}, \{\tau(t)\})$ be the schedule corresponding to operator f which has $x(0) = \bigcup_{i=1}^{p} x_i(0)$ as global initial state. $\gamma(t)$ is a subset of $\{1, 2, \ldots, p\}$ and $i \in \gamma(t)$ means that program P_i has completely finished executing an iteration between time $t-1$ and time t. In other words, if k was the iteration of program P_i at time $t-1$, at time t the iteration number must be $k+1$. The value $\tau(t)$ is an element of $\mathcal{N}^{p \times p}$ satisfying $0 \leq \tau_i^j < t$, for all $t = 1, 2, \ldots$ and $i, j = 1, 2, \ldots, p$ ([39]). $\tau_i^j(t)$ simply returns to program P_i the iteration number of program P_j at time unit t.

Let $\psi(t) = \bigcup_{i=1}^{p} \psi_i(t)$ be the global state vector of the parallel procedure CP, $x(t) = \bigcup_{i=1}^{p} x_i(t)$ the set of p solutions at any point t, and $h(t) = \bigcup_{i=1}^{p} h_i(t)$ a *global virtual data structure* representing the state of all the information gathered by the sequential programs of procedure CP. The global state vector $\psi(t)$ is represented by the constituents $h(t)$ and $x(t)$ of the procedure CP, which is given by $\psi(t) = h(t) \bigcup x(t)$ at any point in time t. If no information is exchanged among the search programs, the state vector $\psi(t)$ of the parallel procedure CP can be found by executing the following iterative maps:

$$x_i(t+1) = \begin{cases} x_i(t) & \text{if } i \notin \gamma(t) \\ s_i(N(x_i(\tau_i^i(t))), h_i(\tau_i^i(t))) & \text{if } i \in \gamma(t) \end{cases} \tag{6}$$

$$h_i(t+1) = \begin{cases} h_i(t) & \text{if } i \notin \gamma(t) \\ \upsilon_i(x_i(\tau_i^i(t+1)), h_i(\tau_i^i(t))) & \text{if } i \in \gamma(t) \end{cases} \tag{7}$$

Here $\tau_i^i(t)$ returns the iteration number of program P_i at time unit t. The entry $x_i(\tau_i^i(t))$ represents the current solution x_i of program P_i at iteration $\tau_i^i(t)$ and $N(x_i(\tau_i^i(t))$ is the neighborhood of the current solution x_i. In general, the entry $h_j(\tau_i^j(t))$ represents the state of memories h_j at iteration $\tau_i^j(t)$ of program P_j, which are used by program P_i to find $h_i(k+1)$. When there is no communication between search programs, program P_i only uses its own tabu memories. Notice that the set of equations (6) and (7) does not represent an interacting system as understood in this paper, the state $\psi_i(t)$ only depends on the internal dynamic of subsystem f_i.

4.4 Iterative map of cooperative search procedures

To model the evolution of a truly cooperative procedure, one has to factor in the interactions among the subsystems. In tabu search, these interactions may take the form of sharing the contents of the tabu memories among the sequential search programs. Therefore, equation (5) has to be modified.

Let $\phi_i \subset \{1, 2, \ldots, p\}$ be a subset of the indices of the sequential programs $P_1, P_2, \ldots, P_p$. For each sequential program P_i, the new current solution $x_i(k+1)$ obtained from the neighborhood $N(x_i)$ at iteration k (step 2 of a tabu iteration) is given by:

$$x_i(k+1) = s_i(N(x_i(k)), (h_i(k), \{h_j(\tau_i^j(t))\})) \text{ where } j \in \phi_i. \tag{8}$$

The set $\{h_j(\tau_i^j(t))\}$ represents the dependency of program P_i on the content of the tabu memories of programs P_j for $j \in \phi_i$. At iteration k of program P_i, the new current solution $x_i(k+1)$ is selected based on the state $h_i(k)$ of the tabu memories of program P_i and on the state of the shared tabu memories of programs P_j represented by $\bigcup_{(j\in\phi)} h_j(\tau_i^j(t))$. When information is exchanged among the search programs, the global state vector $\psi(t)$ of a cooperative procedure can be found by executing the following iterative maps:

$$x_i(t+1) = \begin{cases} x_i(t) & \text{if } i \notin \gamma(t) \\ s_i(N(x_i(\tau_i^i(t))), (h_i(\tau_i^i(t)), \{h_j(\tau_i^j(t))\})) & \text{if } i \in \gamma(t) \text{ and } j \in \phi_i \end{cases} \tag{9}$$

$$h_i(t+1) = \begin{cases} h_i(t) & \text{if } i \notin \gamma(t) \\ \upsilon_i(x_i(\tau_i^i(t+1)), h_i(\tau_i^i(t))) & \text{if } i \in \gamma(t) \end{cases} \tag{10}$$

The set of equations in (9) and (10) constitutes a discrete-time dynamical system representation of a cooperative search. Each subsystem i has its own internal dynamics given by the search strategy of program P_i, and also interacts with other subsystems by sharing the state $h(t)$.

5 Systemic cooperation

Systemic cooperations are macroscopic phenomena emerging from correlated interactions. Using the model of the previous section, we define in more details how interactions among programs are correlated to each other and how

systems of correlated interactions build up. We first describe how sharing information impacts the components ψ_i (i.e. x_i and/or h_i) of the global state of the system. Then, we show that through the iterations of the global system given by (9) and (10), changes to a single component ψ_i could propagate to other components of the state vector ψ and support correlated interactions. We will then be in the position to derive a definition of systemic cooperation.

To simplify the presentation, we assume in the rest of the paper that information sharing occurs between two programs only, P_i and P_j, where program P_i uses the content of the tabu memories of program P_j to perform steps 1 or 2 of a tabu search iteration. We define $\tilde{h}_i^j(k) = h_i(k) \bigcup h_j(\tau_i^j(t))$ and $\bar{x}_i(k+1) = s_i(N(x_i(k)), \tilde{h}_i^j(k))$, as being the current solution chosen by program P_i at iteration k using gathered information from the data structure h_i at iteration k and from the data structure given by $h_j(\tau_i^j(t))$.

5.1 Effective interaction: impact on a state component ψ_i

Information sharing among search programs, as described in equation (8), does not always have an impact on the search behavior of programs. The following definition focuses on a particular form of information sharing:

Definition 1 *The sharing of information among two programs P_i and P_j is said to be an* **effective interaction** *for program P_i at iteration k if and only if the information obtained by P_i from tabu memories h_j of program P_j causes the selection of a current solution $\bar{x}_i(k+1) = s_i(N(x_i(k)), \tilde{h}_i^j(k))$ different from the solution $x_i(k+1) = s_i(N(x_i(k)), h_i(k))$ that would have been obtained if no interaction had taken place between P_i and P_j.*

Clearly, an effective interaction impacts the search path of program P_i at iteration $k+1$ since $\bar{x}_i(k+1) \neq x_i(k+1)$. We now describe how a single effective interaction at a given iteration k of a program P_i can change the search trajectory and the content of the memories of program P_i for one iteration, several iterations or up to end of the execution of program P_i.

If any of the attributes of solution $\bar{x}_i(k+1)$ is stored among the m fields of $h_i(k+1) = \upsilon_i(\bar{x}_i(k+1), h_i(k))$, then the state of the memory of program P_i at iteration $k+1$ has been modified by the effective interaction at iteration k. Then both components of state $\psi_i(k+1)$, x_i and h_i, have different values following the effective interaction.

Let $\beta_i(k+1) = \bar{x}_i(k+1)$ be the current solution after program P_i has performed an effective interaction with another program at iteration k, $\beta_i(k+2)$ the solution selected by program P_i at iteration $k+1$ from the neighborhood $N(\beta_i(k+1))$ (i.e., $\beta_i(k+2)$ will be the current solution at iteration $k+2$),

and $h_i^{\beta}(k+1)$ the state of the tabu memories of program P_i at the end of iteration $k+1$. Let $\alpha_i(k+1) = x_i(k+1)$ be the current solution of program P_i if no effective interaction had taken place at iteration k, $\alpha_i(k+2)$ the solution selected by program P_i at iteration $k+1$, and $h_i^{\alpha}(k+1)$ the state of the tabu memories of program P_i at the end of iteration $k+1$. There are two conditions that may prevent solution $\beta_i(k+2)$ from being the same as solution $\alpha_i(k+2)$:

(1) either $\alpha_i(k+2) \notin N(\beta_i(k+1))$,
(2) or $\alpha_i(k+2) \in N(\beta_i(k+1))$, but $h_i^{\beta}(k+1) \neq h_i^{\alpha}(k+1)$, causing $\alpha_i(k+2)$ not to be evaluated as the best candidate in the neighborhood $N(\beta_i(k+1))$ according to the information stored in $h_i^{\beta}(k+1)$.

Clearly, if solution $\alpha_i(k+2)$ is not in the neighborhood of the current solution $\beta_i(k+1)$ (condition 1), it cannot be chosen as the new current solution at iteration $k+2$. Condition 2 describes a situation where $\alpha_i(k+2) \in N(\beta_i(k+1))$, but the information is different and results in $\alpha_i(k+2)$ being rejected. Thus, if either condition is true, the selection of the new current solution at iteration $k+2$ will be changed by the effective interaction at iteration k. Memories $h_i^{\beta}(k+2) = \upsilon_i(\bar{x}_i(k+2), h_i^{\beta}(k+1))$ of program P_i will then contains attributes of a solution in the β-path of program P_i. That is, these attributes are different from those that would have been stored if program P_i had visited only solutions in the α-path, i.e. if no effective interaction had occurred at iteration k. Fig. 2 illustrates the three potential outcomes of an effective interaction on the search path of program P_i. In the first case, at iteration k of program P_i, an effective interaction chooses solution $\beta_i(k+1)$ as the current solution for iteration $k+1$, but at iteration $k+2$, the solution $\alpha_i(k+2)$ is in the neighborhood of $\beta_i(k+1)$; the state of the tabu memories has not been modified by the selection of solution $\beta_i(k+1)$ or the fields updated by $\beta_i(k+1)$ had no impact on the choice of the new current solution. The search path returns to its α-course in the solution space, that is, $\beta_i(k+2) = \alpha_i(k+2)$. In this case, the effective interaction changes the path of program P_i at iteration k, but $\beta_i(k+1)$ has no further impact on its search behavior. In the second case, an effective interaction attracts the search path to a new region of the solution space for a few iterations, but eventually resumes its original course after l iterations, that is, $\beta_{i,k+l} = \alpha_{i,k+l}$. In the third case, the interference caused by the effective interaction completely modifies the search behavior of program P_i.

Notice the impact of such effective interactions on the memory contents h_i. At each iteration in the β-path, program P_i updates h_i with attributes of a β-solution. Storing this information in the program memories will either overwrite fields which have been filled with attributes of other β-solutions, or will overwrite fields which have been filled by attributes of solutions visited before entering into the β-path. The content of the tabu memories of program P_i could be completely overwritten by attributes of solutions visited in the

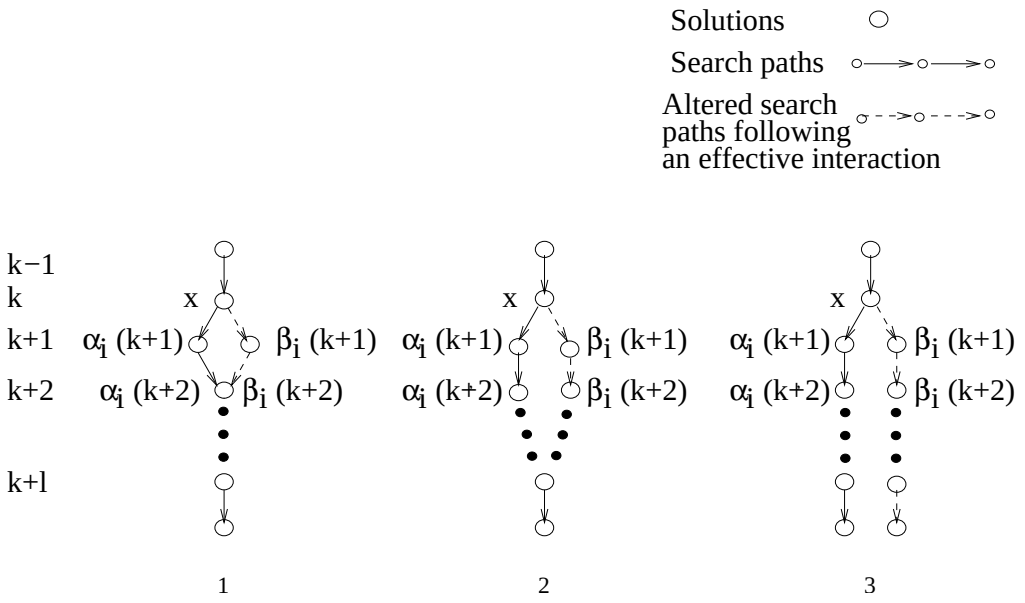


Fig. 2. Potential outcomes following an effective interaction

β-path.

5.2 Propagation in ψ of changes in a component ψ_i

In the previous section, we have seen that an effective interaction deviates the search path of a program. We now proceed to explain how a single effective interaction performed by a program P_i can impact the state of other components $\psi_y,\ y \neq i$, of the dynamical system described by equations (9) and (10). This is the source of the correlated interactions.

We differentiate between two categories of effective interactions:

- Independent effective interactions
- Correlated effective interactions

A program P_i executes an independent effective interaction with a program P_j when the state of the tabu memories of program P_j is given by equation (7), i.e., the memories of program P_j are not modified by the information exchange. We have a correlated effective interaction otherwise. Unless explicitly mentioned, for the remainder of this paper, we consider only effective interactions and refer to interactions as either "independent interactions" or "correlated interactions", or simply as "interactions" when the type of interaction is irrelevant to the context.

Assume that program P_i executes an independent interaction with P_j at time $t - l$. Let also assume that in the interval $[t - l, t]$, program P_i executes a

β-path and that it performs no other independent or correlated interaction. Let $k_i = \tau_i^i(t-l)$ be the iteration of program P_i at time $t-l$. Let also assume that the state $\psi_i(k_i)$ can be obtained by running the iterative map $\psi_i(k+1) = f_i(\psi_i(k)),\ k = 0, 1, \ldots k_i - 1$ (eq. 5), i.e. the state $\psi_i(k_i)$ is as if program P_i had been executed without any interaction with other programs. Let $\tilde{h}_i^j(k_i) = h_i(k_i) \bigcup h_j(\tau_i^j(t-l))$ be the information that will impact the choice of the first solution in the β-path of program P_i in the interval $[t-l, t]$. This first solution is given by:

$$\bar{x}_i(k_i+1) = s_i(N(x_i(k_i)), \tilde{h}_i^j(k_i)). \tag{11}$$

Similarly, let us assume that a program P_y, which explores the solution space concurrently with P_i, executes an correlated interaction with program P_i at time $t-l+\Delta$, $\Delta < l$. Let us further assume that in the time interval $[t-l+\Delta, t]$ program P_y performs no other effective interaction. The value of Δ must be large enough to allow the memories of program P_i to gather information about solutions in the β-path of program P_i; otherwise, the interaction of P_y with P_i will be an independent interaction. Let $k_y = \tau_y^y(t-l+\Delta)$ be the iteration of program P_y at $t-l+\Delta$, and let $\tilde{h}_y^i(k_y) = h_y(k_y) \bigcup h_i(\tau_y^i(t-l+\Delta))$ be the union of the two memory components that will be used to determine the next current solution of program P_y following the correlated interaction. This first solution is given by:

$$\bar{x}_y(k_y+1) = s_y(N(x_y(k_y)), \tilde{h}_y^i(k_y)) \tag{12}$$

which is also a β-solution.

Technically, these two interactions are similar. Both initiate a β-path after an effective interaction (Equations 11 and 12). However there exists an important difference between these two interactions. The occurrence of the independent interaction between programs P_i and P_j, the current solution $x_i(k_i)$ and the gathered information $\tilde{h}_i^j(k_i)$ are fully determined by the initial conditions $x_i(0)$ and $x_j(0)$ of the two programs, their dynamics f_i and f_j, as well as the rules of interactions as defined in section 4.4. This is not the case for the correlated interaction between programs P_y and P_i. The occurrence of this interaction is caused in part by the emergence of a β-path in program P_i. Gathered information $\tilde{h}_y^i(k_y)$ contains information about the solutions found in the β-path of P_i rather than information about solutions visited following the search strategies of programs P_i and P_y. Correlated interactions contribute to distance the search pattern of cooperating programs from the search strategy of the individual programs forming the cooperative procedure.

According to our dynamical system representation of cooperative search, in an independent interaction, the component ψ_i of the global state ψ is modified,

which does impact the dynamics of the subsystem f_i (such impact is defined above as a β-path). This in turn, may generate interactions with other subsystems, which will then propagate modifications in component ψ_i to other components of the global state vector ψ. This corresponds to the dynamics of systemic cooperation:

Definition 2 *A systemic cooperation is a set B of β-paths in an interval $[t-l,t]$ such that, for any β-path $\in B$ and any point in the interval $[t-l,t]$, the iterative maps of equations (9)and (10) involve only β-solutions and β-content from β-paths $\in B$.*

Systemic cooperations are "systems" of interactions, i.e., they are interactions that occur because of the occurrence of other interactions. Since these interactions are effective interactions, each one generates a β-path, changing the search pattern of one of the programs involved in the interaction. As the interactions in systemic cooperations are correlated, systemic cooperations create control dependencies among the cooperating programs. Furthermore, as seen for program P_y above, these control dependencies do not find their origin in any of the specific sequential search strategy associated to the cooperating search programs. Rather, these control actions leading to the β-path in P_y, come from a mix of α/β-paths and α/β-content of memories of three subsystems ψ_i, ψ_j and ψ_y. In this context, systemic cooperations can be viewed as yielding emergent macroscopic search strategies for the exploration of the solution space. The signature of these macroscopic behaviors can be found in the data gathered in the constituents h_i of the subsystems involved in the various systemic cooperations. One can also study and characterize systemic cooperations through the evolution of the global virtual memory of the discrete-time dynamical system ψ. In the next section, we demonstrate the existence and impact of systemic cooperations through the analysis of constituents h_i.

6 Experimentation

In this section we describe simulations that have been designed for the purpose of having a high level of correlated interactions. The simulations used two different cooperative procedures CP and $\overline{CP}$. Each cooperative procedure runs several identical instantiations of the tabu search program described in [9], each instantiation based on different search parameters. The cooperative procedures solved the location-allocation problem also addressed in [9]. This optimization problem has two types of decision variables: design variables, which are Boolean (0/1); and flow variables, which are continuous. The tabu search programs run in the space defined by the design variables; therefore, the solution space X is composed of Boolean vectors of size n, the number of discrete design variables. Tests are performed using a network of Sparc stations, there-

fore the cooperative procedures run on a distributed memory system where gathered information is shared using the message passing paradigm.

6.1 Description of the cooperative procedure CP

Let m_s be a message sent from program P_i to some other program, m_r a message received by program P_i from some other program, and m_0 a special empty message that can be sent and received by any program. Let $h_i(k)$ contain the value of the best solution found by program P_i at iteration k. The cooperating tabu programs are run synchronously, therefore the value k of the tabu iterations is a valid index for the discrete steps of the global state of procedure CP. Information available for sharing is exchanged synchronously among programs at each tabu iteration. A message m_s is sent by program P_i at iteration $k+1$ containing $x_i(k+1)$ if $x_i(k+1) < x_i(k)$; otherwise, P_i has no valuable information to share and message m_0 is sent.

Each program P_i executes the following operations at each iteration k:

- Find α, the best solution in the neighborhood $N(x)$ of the current solution,
- Compare α with β, where β is the content of the message m_r received at iteration $k-1$,
- Choose the best solution among α and β as the current solution for iteration $k+1$.

If the content of message m_r is chosen, then an interaction occurs for program P_i. Otherwise, the program proceeds according to information gathered in its own tabu memories.

The communication structure indicates between which programs exchanges of information take place at each iteration. In the case of procedure CP, at each iteration a program P_i sends its message m_s to only one other program according to the following communication structure: $C(P_i, k) = P_{(i+k) \bmod p}$, where P_i is the source program, $P_{(i+k) \bmod p}$ is the destination program of the message sent by program P_i at iteration k, and p is the number of cooperating programs in the cooperative procedure. Special provision is made for the case where $P_{(i+k) \bmod p} = P_i$ (a program cannot send a message to itself).

To facilitate the comparison of the experimental results, each solution and message were mapped to a set of integers ranging from 1 to 800 on a first-come, first-served basis. Consequently, the values in the following tables indicate only if solutions and messages are different; they do not refer to actual values of solutions to the location-allocation problem.

Table 1
The cooperative procedure CP using four programs

	Programs 1 to 4				Program P_1			
iter	P_1	P_2	P_3	P_4	$x(k)$	$x(k+1)$	m_s	m_r
1	1	2	3	4	1	5	$\emptyset$	$\emptyset$
2	5	6	7	8	5	9	$\emptyset$	6
3	6	7	8	12	6	13	6	$\emptyset$
4	13	6	7	8	13	17	13	7
5	7	8	13	6	7	14	7	$\emptyset$
6	14	21	7	8	14	24	14	$\emptyset$
7	24	14	21	27	24	28	24	$\emptyset$
8	28	21	24	31	28	32	$\emptyset$	$\emptyset$
9	32	33	7	8	32	36	32	$\emptyset$
10	36	32	38	39	36	40	36	$\emptyset$
11	40	21	36	27	40	42	$\emptyset$	$\emptyset$
12	36	43	44	45	36	46	$\emptyset$	$\emptyset$
13	46	15	47	48	46	49	$\emptyset$	$\emptyset$
14	49	50	38	15	49	52	49	$\emptyset$
15	52	53	26	49	52	54	52	$\emptyset$
16	54	52	53	27	54	56	$\emptyset$	$\emptyset$
17	56	57	7	8	56	59	$\emptyset$	57
18	57	60	61	12	57	53	57	12
19	12	57	63	61	12	65	$\emptyset$	63
20	63	61	67	68	63	69	$\emptyset$	$\emptyset$
21	57	67	57	61	57	53	$\emptyset$	$\emptyset$
22	53	71	72	57	53	60	$\emptyset$	$\emptyset$
23	60	74	57	71	60	66	$\emptyset$	$\emptyset$
24	66	76	71	76	66	53	$\emptyset$	76
25	76	77	74	78	76	60	$\emptyset$	74
26	74	79	80	74	74	62	$\emptyset$	$\emptyset$
27	62	76	81	76	62	82	$\emptyset$	$\emptyset$
28	82	83	74	85	82	66	$\emptyset$	$\emptyset$
29	57	74	86	71	57	67	$\emptyset$	$\emptyset$
30	67	76	71	74	67	87	67	$\emptyset$
31	87	67	78	89	87	90	$\emptyset$	$\emptyset$
32	67	71	77	77	67	92	$\emptyset$	$\emptyset$
33	92	77	93	76	92	94	$\emptyset$	$\emptyset$
34	94	95	71	74	94	96	$\emptyset$	71
35	67	71	76	89	67	97	$\emptyset$	$\emptyset$
36	97	79	74	79	97	98	97	$\emptyset$
37	98	97	89	76	98	99	$\emptyset$	$\emptyset$
38	99	57	79	100	99	101	$\emptyset$	$\emptyset$
39	101	74	76	71	101	67	$\emptyset$	$\emptyset$
40	67	76	74	74	67	98	$\emptyset$	$\emptyset$
41	98	79	89	80	98	99	$\emptyset$	$\emptyset$
42	99	74	77	71	99	103	$\emptyset$	$\emptyset$
43	103	71	101	79	103	98	$\emptyset$	$\emptyset$
44	98	97	71	74	98	105	$\emptyset$	97
45	97	104	79	57	97	106	$\emptyset$	$\emptyset$
46	97	107	85	76	97	98	$\emptyset$	$\emptyset$
47	98	108	71	107	98	110	$\emptyset$	$\emptyset$
48	110	111	74	89	110	112	$\emptyset$	$\emptyset$
49	97	113	111	77	97	99	$\emptyset$	$\emptyset$
50	99	111	77	71	99	98	$\emptyset$	$\emptyset$

6.2 Description of the cooperative procedure $\overline{CP}$

The cooperative procedure $\overline{CP}$ is a modified version of CP. $\overline{CP}$ is not a fully cooperative procedure; the program P_1 has been altered such that it cannot read the messages received from other programs. Program P_1 never executes an interaction with other programs. The search path of P_1 is defined by equations (6) and (7), it is an α-path only. Notice, however, that although P_1 does not read its messages, it still continues to send messages to other programs using the same communication structure as defined for procedure CP. Consequently, the other programs can still realize interactions based on information received from P_1.

6.3 Impact of shared gathered information on P_1

Table 1 displays the search path of four tabu search programs for the first 50 iterations (columns 2 to 5). Each program uses a different search strategy and shares information according to the rules defined for CP. The next four columns of Table 1 focus on the search program P_1 and show the update pattern of variables $x(k)$ and $x(k+1)$, as well as the messages sent (m_s) and received (m_r) by P_1 at each iteration. Table 2 presents similar results for the simulation realized with $\overline{CP}$.

Columns $x(k)$ in Table 1 and 2 show the search path of program P_1 for the cooperative procedures CP and $\overline{CP}$, respectively. In $\overline{CP}$, the program P_1 executes an α-path for all iterations, since it does not perform any interaction. On the other hand, 8 interactions are performed by program P_1 during the 50 iterations of CP. Table 1 shows clearly that interactions, based on gathered information from P_2, P_3 and P_4, have a significant impact on the search behavior of program P_1.

Given that $P_1 \in CP$ has executed several β-paths, this validates the prediction of the model presented in Section 4: attributes from β-solutions are different from that of $\overline{CP}$. The columns of sent messages in Tables 1 and 2 show that the two sets of sent messages are totally different. Together, columns $x(k)$ and m_s in Tables 1 and 2 show that interactions have a strong influence on both the search path and the information made available for sharing.

6.4 Reading systemic cooperations from the experiments

The values m_r in Tables 1 and 2 show the existence of a transfer of β-solution attributes among β-paths of the cooperative procedures; therefore, illustrating

Table 2
The cooperative procedure $\overline{CP}$ using four programs

	Programs 1 to 4				Program P_1			
iter	P_1	P_2	P_3	P_4	$x(k)$	$x(k+1)$	m_s	m_r
1	1	2	3	4	1	5	0	0
2	5	6	7	8	5	9	$\emptyset$	6
3	9	7	8	12	9	202	9	$\emptyset$
4	202	9	7	8	202	203	202	7
5	203	8	202	9	203	205	203	$\emptyset$
6	205	206	7	8	205	209	205	$\emptyset$
7	209	205	206	51	209	210	209	$\emptyset$
8	210	211	209	205	210	214	210	$\emptyset$
9	214	205	7	8	214	218	214	$\emptyset$
10	218	214	220	58	218	221	$\emptyset$	$\emptyset$
11	221	222	47	223	221	224	221	$\emptyset$
12	224	205	38	221	224	226	224	$\emptyset$
13	226	224	228	229	226	230	226	$\emptyset$
14	230	219	226	27	230	231	$\emptyset$	$\emptyset$
15	226	232	220	51	226	233	$\emptyset$	$\emptyset$
16	233	234	38	45	233	235	$\emptyset$	$\emptyset$
17	235	217	7	8	235	236	$\emptyset$	$\emptyset$
18	226	227	237	238	226	239	$\emptyset$	238
19	239	219	240	237	239	242	$\emptyset$	240
20	242	205	243	244	242	233	$\emptyset$	$\emptyset$
21	233	243	245	237	233	247	$\emptyset$	$\emptyset$
22	247	248	243	250	247	239	$\emptyset$	$\emptyset$
23	239	251	252	248	239	242	$\emptyset$	251
24	242	252	255	256	242	233	$\emptyset$	256
25	233	251	258	255	233	247	$\emptyset$	$\emptyset$
26	226	260	261	262	226	230	$\emptyset$	$\emptyset$
27	230	263	255	256	230	266	230	$\emptyset$
28	266	230	263	269	266	270	266	263
29	270	269	266	272	270	273	$\emptyset$	$\emptyset$
30	266	263	274	275	266	276	$\emptyset$	$\emptyset$
31	276	277	278	279	276	280	$\emptyset$	278
32	280	267	281	282	280	283	$\emptyset$	$\emptyset$
33	266	260	282	285	266	286	$\emptyset$	$\emptyset$
34	286	277	263	279	286	276	$\emptyset$	$\emptyset$
35	276	278	287	288	276	289	$\emptyset$	278
36	289	281	290	291	289	292	289	$\emptyset$
37	292	289	281	285	292	293	292	$\emptyset$
38	293	263	292	279	293	294	$\emptyset$	$\emptyset$
39	294	287	263	288	294	295	$\emptyset$	$\emptyset$
40	295	281	296	291	295	289	$\emptyset$	$\emptyset$
41	289	284	278	298	289	293	$\emptyset$	$\emptyset$
42	293	263	284	282	293	294	$\emptyset$	$\emptyset$
43	294	290	300	285	294	295	$\emptyset$	$\emptyset$
44	295	301	278	303	295	289	$\emptyset$	$\emptyset$
45	289	278	287	282	289	306	$\emptyset$	$\emptyset$
46	306	281	281	279	306	307	306	$\emptyset$
47	307	308	306	285	307	309	$\emptyset$	308
48	306	310	263	311	306	312	$\emptyset$	$\emptyset$
49	312	313	290	288	312	314	$\emptyset$	$\emptyset$
50	314	308	281	279	314	306	$\emptyset$	$\emptyset$

the existence of systemic cooperation structures among these cooperating programs. The variable m_r represents the messages received by program P_1 from the other programs. Recall that procedures CP and $\overline{CP}$ are identical, except for the reading of the messages by program P_1. Therefore, the differences in the search behavior of program P_1 are due to the fact that P_1 in CP reads its messages and performs interactions.

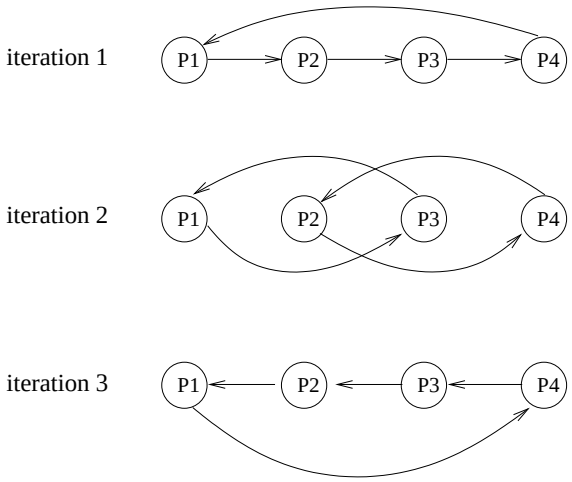
By definition, a program P_i performs an independent interaction if it uses α-solution attributes from program P_j to determine the next current solution in its search path. If there were only independent interactions in the cooperative procedure CP, changes to the search behavior of program P_1 should not influence the search behavior of the three other programs P_2, P_3 and P_4. Programs P_2, P_3 and P_4 would pick-up only α-solution attributes from P_1 and there would be no systemic cooperation. Yet, as can be observed by comparing Tables 1 and 2, programs P_2, P_3 and P_4 do not explore the same regions of the solution space in the two cooperative procedures, except for the first three iterations. There is a dramatic difference in the search behavior of the sequential programs between the two cooperative procedures. The only possible way for these differences to occur, is if some β-solution attributes from P_1 are picked-up by programs P_2, P_3 and P_4 once P_1 has started to read its messages. Therefore, correlated interactions take place among cooperating programs in CP.

The existence of correlated interactions supports the premise of systemic cooperations in CP. These systemic cooperations include at least two interactions (one performed by P_1 and one executed by either P_2, P_3 and P_4 using β-solutions attributes from P_1). Observing the results of these two cooperative procedures, one can trace systemic cooperation structures that are in fact more complex. For example, column m_r is different for CP and $\overline{CP}$. The only way for this to happen, is for at least one program among P_i, $i \neq 1$, to have passed some of its β-values to P_1; thereby, creating a sequence of at least three consecutive interactions. In general, as observed in [37], the structure of systemic cooperations is not limited to sequences of different lengths of interactions. They can take all kinds of more or less regular shapes. However, as explained in the next section, the system control parameters have a strong impact on the structure of systemic cooperations.

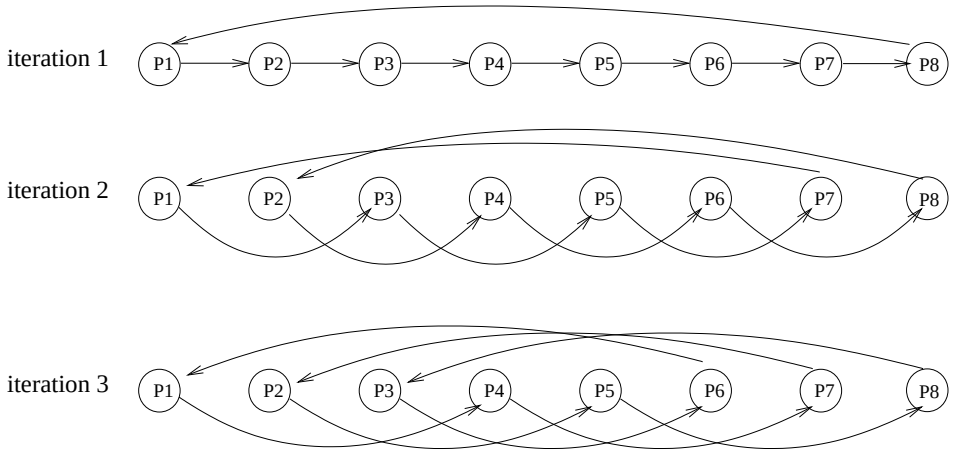
6.5 System control parameters & systemic cooperation

The design of any cooperative algorithm must define the number of sequential programs, the components of the gathered information made available for sharing, the frequency of the sharing and the programs among which information is to be shared. In this section, we show briefly that systemic cooperations

and the system control parameters interact directly in a complex manner with each other.



(a)



(b)

Fig. 3. Communication structure with: (a) 4 programs; (b) 8 programs

For example, it is obvious that the logical topology of interconnections among the sequential programs has a direct impact on the structure of the systemic cooperations. Interactions can only take place among search programs which share information. Each topology then causes program to have a different interaction pattern with the other programs, which in turn generates different interactions.

A similar situation happens when the number of cooperating programs is modified. Applying the interconnection topology defined in Section 6.1 ($C(P_i, k) =$

Table 3
Cooperative procedure CP with 5 programs

	Programs 1 to 5					Program P_1			
iter	P_1	P_2	P_3	P_4	P_5	$x(k)$	$x'(k)$	m_s	m_r
1	1	2	3	4	587	1	5	$\emptyset$	$\emptyset$
2	5	6	7	8	588	5	9	$\emptyset$	7
3	7	8	588	12	6	7	590	7	8
4	8	588	592	6	7	8	12	8	$\emptyset$
5	12	8	588	592	6	12	455	12	$\emptyset$
6	455	238	12	8	598	455	599	$\emptyset$	$\emptyset$
7	12	8	601	602	603	12	604	12	$\emptyset$
8	604	45	12	8	6	604	608	$\emptyset$	$\emptyset$
9	12	609	610	39	611	12	610	$\emptyset$	$\emptyset$
10	610	8	612	613	614	610	612	$\emptyset$	$\emptyset$
11	612	39	12	8	617	612	615	$\emptyset$	$\emptyset$
12	12	45	618	27	619	12	65	$\emptyset$	$\emptyset$
13	65	620	610	39	598	65	610	$\emptyset$	$\emptyset$
14	610	58	621	45	611	610	622	$\emptyset$	$\emptyset$
15	622	39	612	620	623	622	612	$\emptyset$	$\emptyset$
16	612	45	618	58	6	612	65	$\emptyset$	$\emptyset$
17	65	620	610	39	10	65	610	$\emptyset$	$\emptyset$
18	610	8	621	45	624	610	622	$\emptyset$	$\emptyset$
19	622	53	12	8	625	622	612	$\emptyset$	53
20	12	421	626	625	627	12	8	$\emptyset$	627
21	627	53	629	626	625	627	632	627	$\emptyset$
22	632	625	626	625	624	632	635	$\emptyset$	$\emptyset$
23	635	636	637	636	625	635	639	635	$\emptyset$
24	639	53	641	640	635	639	642	$\emptyset$	$\emptyset$
25	642	57	626	625	645	642	646	642	$\emptyset$
26	646	60	642	57	625	646	650	$\emptyset$	642
27	642	66	651	636	652	642	651	$\emptyset$	$\emptyset$
28	651	651	653	654	636	651	655	$\emptyset$	$\emptyset$
29	655	480	655	656	657	655	658	$\emptyset$	$\emptyset$
30	642	60	658	648	655	642	659	$\emptyset$	$\emptyset$
31	659	66	660	636	648	659	651	$\emptyset$	$\emptyset$
32	651	660	661	654	636	651	662	$\emptyset$	$\emptyset$
33	662	478	663	625	657	662	664	662	$\emptyset$
34	664	499	662	666	625	664	667	$\emptyset$	$\emptyset$
35	667	57	669	602	666	667	642	$\emptyset$	$\emptyset$
36	642	60	663	666	671	642	672	$\emptyset$	$\emptyset$
37	672	82	661	673	666	672	664	$\emptyset$	$\emptyset$
38	664	57	676	666	602	664	678	$\emptyset$	$\emptyset$
39	678	66	679	680	666	678	681	678	$\emptyset$
40	681	60	682	683	678	681	685	$\emptyset$	$\emptyset$
41	685	478	642	666	685	685	686	$\emptyset$	$\emptyset$
42	686	66	687	680	666	686	689	$\emptyset$	$\emptyset$
43	689	60	642	691	683	689	681	$\emptyset$	$\emptyset$
44	681	478	651	683	684	681	685	$\emptyset$	$\emptyset$
45	685	66	655	692	680	685	686	$\emptyset$	$\emptyset$
46	686	57	642	680	691	686	689	$\emptyset$	$\emptyset$
47	689	72	659	691	684	689	687	$\emptyset$	72
48	72	75	651	683	680	72	693	$\emptyset$	$\emptyset$
49	678	72	662	666	691	678	694	$\emptyset$	$\emptyset$
50	694	695	664	696	666	694	697	$\emptyset$	$\emptyset$

$P_{(i+k) \bmod p}$), fig. 3.a shows between which programs interactions take place for the first three iterations of the cooperative procedure CP. Fig. 3.b, show those interactions once the number of programs goes from 4 to 8 while keeping the same interconnection model. At iteration 1, program P_1 reads the memory component of program P_8 rather than program P_4 in Fig. 3.a. Therefore, program P_1 exchanges gathered information with program P_8. At iteration 2, P_2 reads the memory component of program P_7 rather than P_3, etc. By changing the number of programs, one modifies the dependencies among the search programs resulting in different interactions in the procedure with 8 programs when compared with the procedure with 4 search programs. It is always the case that adding new search programs impacts on the structure of systemic cooperations, unless the new programs are not allowed to interact at all with the previous programs, but of course then there is no cooperation!

We have run a simulation to illustrate the impact of the number of programs on the search behavior and occurrence of interactions. Table 3 reports the results of this simulation which uses the cooperative procedure CP with 5 programs rather than 4. Again, we can observe significant modifications to the search behavior of the sequential programs. Comparing Table 3 and Table 1, one sees that new gathered information is made available because of the exploration of the solution space performed by the new sequential program P_4. One can also observe changes in the set of interactions that results from the modifications to the dependencies among the sequential programs: the interactions are not the same for program P_1 in the experiment.

7 Conclusions & discussion

Cooperative search algorithms are parallel (distributed) search methods that combine several individual programs in a single search system. The programs cooperate with each other by sharing attributes of already visited solutions. In general, the search pattern of an individual search program depends on its search strategy, the initial solution and on the values of its search parameters. By extension, the behavior of cooperative search procedures is often characterized in the same way. Thus, for example, it is assumed that programs exchange information from search histories undisturbed by cooperation, that interactions have only a limited and local impact on cooperating programs. This gave rise to the belief that the search strategy and search parameters of the individual programs fully control the behavior of cooperative procedures.

In this paper, we have described systemic cooperations, which are macroscopic phenomena occurring in cooperative search procedures. Systemic cooperations create complex control dependencies among programs as exemplified by the description of their dynamics: A first independent interaction causes one pro-

gram to adopt a new search pattern and to store attributes of the combined search history of both interacting programs. Subsequent correlated interactions amplify this movement by combining search patterns that could not have been generated by any of the search strategy of the individual programs. Correlated interactions also contribute to store information which does not correspond to any specific search history. Consequently, the control actions implemented by systemic cooperations are not based on any specific search strategy or search history of individual sequential search programs; rather they are the manifestation of an emergent search strategy evolves from the interactions among the search programs. In turn, this emergent search strategy has a tremendous influence on the search behavior of cooperative procedures; but it is unlikely that these algorithmically unspecified macroscopic behaviors of cooperative procedures always automatically favor convergence toward good regions of the solution space.

We think that future designs of cooperative algorithms must include features in the individual search programs which specifically address and control the manifestations of macroscopic behaviors of cooperative procedures. Unfortunately, to this day, the use of the emergent behavior of distributed computation to realized specific algorithmic functionalities is a rather uncharted area in the field of algorithm design. At this point, we have used only simple approaches to find better designs for cooperative algorithms. Exploiting the relations existing between systemic cooperations and the system control parameters, we have seek to eliminate or reduce the impact of emergent behaviors that didn't seem to be useful. For example, we have identified noisy dynamics created by self-referential loops among cooperating programs. To break these self-referential loops, in [32,38] we have experimented with cooperative procedures combined with multilevel search algorithms. The multilevel concept introduces a hierarchy among the search programs. Information can only be shared among adjacent levels in the hierarchy, which yields a very simple communication structure: program P_i can only interact with programs P_{i-1} and P_{i+1}. Correlated interactions can only propagate between levels, which makes more tractable the impact of the spontaneous dynamics emerging in the system. Moreover, in this design, interactions are not allowed to change the search path directly. Rather, they change the neighborhood structure (the type of moves) executed by the search method, which also helps to control self-referential loops. This seems to yield less brittle procedures and lead to more stable and efficient cooperative algorithms. The multilevel cooperative search procedure consistently produced very good solutions for all problem instances.

We think that simplified design of cooperative search procedures like this multilevel algorithm above, can help to discover spontaneously occurring macroscopic behaviors and to harness their power to improve the performance of the cooperative procedures. A more challenging research avenue is to try to

classify the macroscopic phenomena in categories of operators or programming primitives. Algorithm designers envision computing environments as defining a set of basic operators that can be used to express algorithms. Such a characterization of the computational behavior of different macroscopic phenomena could simplify their use as primitives in the design of cooperative algorithms.

Acknowledgments

Funding for this project has been provided by the Natural Sciences and Engineering Council of Canada through its Research Grant program and by the Fonds F.C.A.R. of the Province of Québec.

References

[1] R. Battiti and G. Tecchiolli. Parallel Biased Search for Combinatorial Optimization: Genetic Algorithms and TABU. *Microprocessors and Microsystems*, 16(7):351–367, 1992.

[2] D. Chazan and W. Miranker. Chaotic Relaxation. *Linear Algebra and its Applications*, 2:199–222, 1969.

[3] S.H. Clearwater, T. Hogg, and B.A. Huberman. Cooperative Problem Solving. In B.A. Huberman, editor, *Computation: The Micro and the Macro View*, pages 33–70. World Scientific, 1992.

[4] S.H. Clearwater, B.A. Huberman, and T. Hogg. Cooperative Solution of Constraint Satisfaction Problems. *Science*, 254:1181–1183, 1991.

[5] J. Cohoon, S. Hedge, W. Martin, and D. Richards. Punctuated Equilibria: A Parallel Genetic Algorithm. In J.J. Grefenstette, editor, *Proceedings of the Second International Conference on Genetic Algorithms and their Applications*, pages 148–154. Lawrence Erlbaum Associates Publishers, 1987.

[6] D. Costa. An Evolutionary Tabu Search Algorithm and the NHL Scheduling Problem. *INFOR*, 33(3):161–178, 1995.

[7] T. G. Crainic and M. Toulouse. Parallel Strategies for Metaheuristics. In F. Glover and G. Kochenberger, editors, *State-of-the-Art Handbook in Metaheuristics*. Kluwer Academic Publishers, 2002.

[8] T.G. Crainic and M. Gendreau. Cooperative Parallel Tabu Search for Capacitated Network Design. *Journal of Heuristics*, 2002. to appear.

[9] T.G. Crainic, M. Gendreau, P. Soriano, and M. Toulouse. A Tabu Search Procedure for Multicommodity Location/Allocation with Balancing Requirements. *Annals of Operations Research*, 41:359–383, 1993.

[10] T.G. Crainic, M. Toulouse, and M. Gendreau. Parallel Asynchronous Tabu Search for Multicommodity Location-Allocation with Balancing Requirements. *Annals of Operations Research*, 63:277–299, 1995.

[11] T.G. Crainic, M. Toulouse, and M. Gendreau. Synchronous Tabu Search Parallelization Strategies for Multicommodity Location-Allocation with Balancing Requirements. *OR Spektrum*, 17(2/3):113–123, 1995.

[12] J.D. Donnelly. Periodic Chaotic Relaxation. *Linear Algebra and its Applications*, 4:117–128, 1971.

[13] B. Gendron and T.G. Crainic. Parallel Branch-and-Bound Algorithms: Survey and Synthesis. *Operations Research*, 42(6):1042–1066, 1994.

[14] F. Glover. Tabu Search – Part I. *ORSA Journal on Computing*, 1(3):190–206, 1989.

[15] F. Glover. Tabu Search – Part II. *ORSA Journal on Computing*, 2(1):4–32, 1990.

[16] V. Gordon, D. Whitley, and A. Bohm. Dataflow Parallelism in Genetic Algorithms. In R. Männer and B. Manderick, editors, *Parallel Problem Solving from Nature II*, pages 533–542. Elsevier Science Publishers B.V., 1992.

[17] V. S. Gordon and D. Whitley. Serial and Parallel Genetic Algorithms as Function Optimizers. In Stephanie Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 177–183. Morgan Kaufmann Publishers, 1993.

[18] D.R. Greening. Parallel Simulated Annealing Techniques. *Physica D*, 42:293–306, 1990.

[19] T. Hogg and C. Williams. Solving the Really Hard Problems with Cooperative Search. In *Proceedings of the 11th National Conference on Artificial intelligence (AAAI93)*, pages 231–236. AAAI Press, 1993.

[20] B.A. Huberman. The Performance of Cooperative Processes. *Physica D*, 42:38–47, 1990.

[21] P. Jog and D Gucht. Parallelisation of Probabilisitc Sequential Search Algorithms. In J.J. Grefenstette, editor, *Proceedings of the Second International Conference on Genetic Algorithms and their Applications*, pages 170–176. Lawrence Erlbaum Associates Publishers, 1987.

[22] P. S. Laursen. Problem-Independent Parallel Simulated Annealing Using Selection and Migration. *Lecture Notes in Computer Science*, 866:408–417, 1994.

[23] A. Le Bouthillier and T.G. Crainic. Cooperative Parallel Method for Vehicle Routing Problems with Time Windows. Publication, Centre de recherche sur les transports, Université de Montréal, Montréal, QC, Canada, 2002.

[24] K-G. Lee and S-Y. Lee. Efficient Parallelization of Simulated Annealing using Multiple Markov Chains: An Application to Graph Partitioning. In Trevor N. Mudge, editor, *Proc. 1992 of the Int. Conf. on Parallel Processing*, pages III 177–180. CRC Press, 1992.

[25] S. W. Mahfoud and D. E. Goldberg. Parallel Recombinative Simulated Annealing: a Genetic Algorithm. *Parallel Computing*, 21:1–28, 1995.

[26] B. Manderick and P. Spiessens. Fine-Grained Parallel Genetic Algorithm. In J. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 416–421. Morgan Kaufmann Publishers, 1989.

[27] P. Moscato. On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms . Publication Report 790, Caltech Concurrent Computation Program, 1989.

[28] P. Moscato. An Introduction to Population Approaches for Optimization and Hierarchical Objective Functions: A Discussion on the role of Tabu Search. *Annals of Operations Research*, 41:85–121, 1993.

[29] P. Moscato and M.G. Norman. A 'Memetic' Approach for the Traveling Salesman Problem. Implementation of a Computational Ecology for Combinatorial Optimization on Message-Passing Systems,. In M. Valero, E. Onate, M. Jane, J.L. Larriba and B. Suarez, editors, *Parallel Computing and Transputer Applications*, pages 187–194. IOS Press, Amsterdam, 1992.

[30] H. Mühlenbein, M. Gorges-Schleuter, and O. Krämer. New Solutions to the Mapping Problem of Parallel Systems - the Evolution Approach. *Parallel Computing*, 6:269–279, 1987.

[31] M. Munetomo, Y. Takai, and Y. Sato. An efficient Migration Scheme for Subpopulation-Based Asynchronous Parallel Genetic Algorithms. In Stephanie Forrest, editor, *Proceedings of the fifth International Conference on Genetic Algorithms*, pages 649–658. Morgan Kaufmann Publishers, 1993.

[32] M. Ouyang, M. Toulouse, K. Thulasiraman, F. Glover, and J.S. Deogun. Multilevel Cooperative Search for the Circuit/Hypergraph Partitioning Problem. *IEEE Transactions on Computer-Aided Design and Integrated Circuits and Systems*, 21(6):685–693, 2002.

[33] C. Pettey, M. Leuze, and J. Grefenstette. A Parallel Genetic Algorithm. In J.J. Grefenstette, editor, *Proceedings of the Second International Conference on Genetic Algorithms and their Applications*, pages 155–161. Lawrence Erlbaum Associates Publishers, 1987.

[34] E. Taillard. Parallel Taboo Search Techniques for the Job Shop Scheduling Problem. *ORSA Journal on Computing*, 6(2):108–117, 1994.

[35] R. Tanese. Parallel Genetic Algorithm for a Hypercube. In J.J. Grefenstette, editor, *Proceedings of the Second International Conference on Genetic Algorithms and their Applications*, pages 177–183. Lawrence Erlbaum Associates Publishers, 1987.

[36] R. Tanese. Distributed Genetic Algorithms. In J.D. Schaffer, editor, *Proc. Third Int. Conference on Genetic Algorithms and their Applications*. Morgan Kaufmann Publishers, 1989.

[37] M. Toulouse, T.G. Crainic, and M. Gendreau. Communication Issues in Designing Cooperative Multi-Thread Parallel Searches. In I.H. Osman and J.P. Kelly, editors, *Meta-Heuristics: Theory and Applications*, pages 501–522. Kluwer Academic Publishers, 1996.

[38] M. Toulouse, K. Thulasiram, and F. Glover. Multi-Level Cooperative Search. In *5th International Euro-Par Parallel Processing Conference, volume 1685 of Lecture notes in Computer Science*, pages 533–542. Springer-Verlag, 1999.

[39] A. Üresin and M. Dubois. Asynchronous Iterative Algorithms: Models and Convergence. In L. Kronsjö and D. Shumsheruddin, editors, *Advances in Parallel Algorithms*, pages 302–342. John Wiley & Sons, Inc., 1992.

[40] M.G.A. Verhoeven and E.H.L. Aarts. Parallel Local Search. *Journal of Heuristics*, 1(1):43–65, 1995.

[41] M. Yannakakis. Computational Complexity. In E. Aarts and J.K. Lenstra, editors, *Local Search in Combinatorial Oprimization*, pages 19–57. John Wiley & Sons Inc., 1997.